CROSSTALK

August 2004 The Journal of Defense Software Engineering Vol. 17 No. 8

Orchestrating a Systems Approach

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
AUG 2004		**00-00-2004 to 00-00-2004**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
CrossTalk. The Journal of Defense Software Engineering. Volume 17, Number 8, August 2004	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **517 SMXS MXDEA,6022 Fir Ave,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**32**	

Systems Approach

ON THE COVER

Cover Design by Kent Bingham.
Cover Photo © Image 100 Ltd.

Software Engineering Technology

Departments

CROSSTALK

PUBLISHER	*Tracy Stauder*
ASSOCIATE PUBLISHER	*Elizabeth Starrett*
MANAGING EDITOR	*Pamela Palmer*
ASSOCIATE EDITOR	*Chelene Fortier-Lozancich*
ARTICLE COORDINATOR	*Nicole Kentta*
CREATIVE SERVICES COORDINATOR	*Janna Kay Jensen*
PHONE	(801) 586-0095
FAX	(801) 777-8069
E-MAIL	crosstalk.staff@hill.af.mil
CROSSTALK ONLINE	www.stsc.hill.af.mil/crosstalk

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 17.

Ogden ALC/MASE
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

Reprints and Permissions: Requests for reprints must be requested from the author or the copyright holder. Please coordinate your request with CROSSTALK.

Trademarks and Endorsements: This DoD journal is an authorized publication for members of the Department of Defense. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the government, the Department of Defense, or the Software Technology Support Center. All product names referenced in this issue are trademarks of their companies.

Coming Events: We often list conferences, seminars, symposiums, etc. that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CROSSTALK Editorial Department.

STSC Online Services: www.stsc.hill.af.mil
Call (801) 777-7026, e-mail: stsc.webmaster@hill.af.mil

Back Issues Available: The STSC sometimes has extra copies of back issues of CROSSTALK available free of charge.

The Software Technology Support Center was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

Defining Systems

I have heard many different definitions and uses for the term *system* with respect to software. Some consider a system to be a collection of software, while others consider it to be the combination of hardware and software merged to perform functions not possible by the individual parts. Some take a holistic approach, including hardware, software, documentation, communication, cost, quality, and even people in their definitions. Perhaps this myriad of definitions is why we also have so much controversy over what systems engineering truly entails.

Apparently, *system* is in the eye of the beholder. An avionics system often consists of a guidance system, a radar system, a flight control system, a fire control system, and others. However, the avionics system is only one part of an aircraft system. Different stakeholders will draw boundaries around the *system* at different locations. Our authors this month each have their own perspective for system as it relates to the information they are trying to share. However, they do keep in common Webster's definition of system: "a group of interrelated, interacting, or interdependent constituents forming a complex whole."

We begin this month with Ivy Hooks' article, *Managing Requirements for a System of Systems* (SOS). Hooks' concerns go beyond the already-present issue of systems and address SOS. The SOS performs functions not possible by any of the individual systems operating alone; the whole is greater than the sum of the parts. An SOS continually evolves as needs change and newer technologies become available. Such large groups have complex requirements that must be carefully planned and managed.

In *Applying CMMI to Systems Acquisition*, Brian P. Gallagher and Sandy Shrum try to alleviate the difficulties faced by organizations acquiring systems by introducing the Capability Maturity Model® Integration (CMMI®) Acquisition Module (AM). The CMMI-AM is a streamlined version of CMMI best practices that can be implemented to help establish effective acquisition practices within acquisition programs.

Dr. Norman F. Schneidewind discusses the needed revisions to the American Institute of Aeronautics and Astronautics' (AIAA) publication, "AIAA Recommended Practice for Software Reliability (R-013-1992)" in *A Recommended Practice for Software Reliability*. While focusing on software reliability, this document and its proposed revision also consider hardware and ultimately systems characteristics.

Dr. Robert Charette, Laura M. Dwinnell, and John McGarry lead our collection of supporting articles with *Understanding the Roots of Process Performance Failure*. In this article, the authors reference the results of numerous program assessments to provide guidance on how to counteract prevalent program performance issues.

Next, Lawrence Bernstein and Dr. Chandra M. R. Kintala discuss one approach to software fault tolerance in *Software Rejuvenation*. Bernstein and Kintala recommend stopping a software program at opportune intervals as a way of cleaning up the internal state of the system and then restarting it at a known, healthy state to prevent a predicted future failure.

Finally, in *Enterprise Composition,* John Wunder discusses enterprise composition as an approach to enterprise information system architectures and shares how it supports the creation and evolution of large enterprise information system architectures such as the Air Force's Global Combat Support System.

We need to look at our mission as part of the big picture described by John Gilligan, Air Force chief information officer, in his January 2004 CROSSTALK article: "… individual software solutions must be integral to and tightly integrated with all components of a system, or in most cases with the *system of systems*. We need to integrate software into our overall systems engineering processes."

Whether your focus thus far has been on software engineering or systems engineering, there exists a need in our defense community to focus on systems engineering and to understand how software engineering plays a critical role in this interdisciplinary approach.

Elizabeth Starrett
Associate Publisher

Managing Requirements for a System of Systems

Ivy Hooks
Compliance Automation, Inc.

As we encounter more system of systems (SOS) and more complex SOS, we must consider the changes that will be required of our existing processes. For example, requirements management has been focused on a system or a product. This article looks at how the SOS has evolved, including what parts of requirements management apply to the SOS and where the process will need revision. It also discusses the need for dynamic scope for the SOS and more use of standards to interface the systems. Challenges definitely exist in SOS, not just in the Department of Defense but also in every aspect of the networked world. Our existing requirements management process is necessary, but not sufficient for the SOS.

There is much in literature depicting system of systems (SOS) [1]. I will use the characteristics Maier [2] has defined (in boldface below), followed by my summary of each.

1. **Operational Independence of the Individual Systems.** If you decompose the SOS, each component system can still perform independently of the others.
2. **Managerial Independence of the Systems.** Each individual system has its own purpose independent of the others and is managed separately for that purpose.
3. **Geographic Distribution.** Often individual systems are distributed over large geographic areas.
4. **Emergent Behavior.** The SOS performs functions not possible by any of the individual systems operating alone. The reason for developing the SOS is to obtain this unique behavior.
5. **Evolutionary Development.** An SOS is never finished; it continually evolves as needs change and newer technologies become available.

Maier defines an SOS as having all or a majority of these characteristics.

Evolution of SOS

The Past

In the first space systems, we built a system to do all of the functions simultaneously. The responsibility for the system fell under one organization, although the work may have been parceled out to many organizations. There was a central point of control. For example, when the National Aeronautics and Space Administration (NASA) built the Apollo space vehicle 40 years ago, NASA built all elements of the vehicle, its launch pad, and many other ground facilities.

When I toured the NASA Goddard Space Flight Center nearly 20 years ago, I questioned the need for dozens of different data processing systems – one for each satellite program. The person providing the tour had no idea why things were like they were, but I later talked to a NASA headquarters person who explained it very clearly. "Of course fewer systems would be better, but we can't take the risk. If Program A and Program B agree to share a ground data system and then one or the other gets cancelled, the remaining program will not have the funds for its data processing system. To protect against this highly probable scenario, it's every man for himself." This can still happen.

"The SOS scope creates the vision and sets the bounds for what is to be accomplished. Scope includes the need, goals, and objectives for the SOS ... Additionally the SOS scope will need to address all the system-to-system interfaces within the SOS."

The Present

Today, we have a number of existing systems that serve many other systems. These systems, e.g., Telemetry Data Relay Satellite System and the Global Positioning System (GPS), enable multiple new systems to accomplish their mission without reinventing the wheel or duplicating capabilities. Writing requirements to interface to these existing systems is generally straightforward, involving an understanding of what the new system must do to interface to the existing system.

Interfacing to a developing system where its design is evolving even as your design is evolving is much more difficult. In the automotive industry, with many computers under the hood of every vehicle, interfaces are a nightmare. One story I was told involved creating a new dashboard – an SOS comprised of entertainment, car information, temperature control, air bags, etc. The designer for the air bag system noticed that if anyone else sent a particular command on the bus, then the air bag would deploy. "But nobody would ever do that," he said. When the dashboard was assembled and an unsuspecting person moved the temperature control, the air bag deployed.

Managing Requirements

If you have not already, you will probably encounter an SOS in the near future. Although I wrote about managing requirements for single systems [3] without regard to the SOS, the basic principles apply. In fact, the basic activities shown in Table 1 are even more essential for managing an SOS than for a single system. In a single system, management is by a program or project manager. Requirements elicitation is the responsibility of system engineers or analysts who report to the program/project manager. In an SOS, these roles will need to be performed, but will be difficult organizationally. While using standards can benefit almost every system, standards may be essential for a successful SOS.

Strategic planning is essential for SOS development. The overall vision must be defined and embraced. Since an SOS does not have a limited life cycle but continues with the evolution of the SOS, its strategic plan must also evolve. The SOS capabili-

ties must evolve as needs change and new technologies become available.

SOS Scope

In product development, it is essential to identify the scope of the product before writing requirements. It is even more important to define the scope of the SOS before embarking on any aspect of requirements writing. The SOS scope creates the vision and sets the bounds for what is to be accomplished. Scope includes the need, goals, and objectives for the SOS. It also includes operational concepts for all life-cycle phases from the viewpoint of all stakeholders. Scope includes the external drivers, e.g., regulations and external interfaces. Additionally the SOS scope will need to address all the system-to-system interfaces within the SOS.

If we can identify the problem to be solved, then we can determine our need, goals, and objectives for the SOS. An example problem might be to obtain more accurate weather data using new technology in the following example:

- **Need.** Validate using the new technology to increase weather forecast accuracy.
- **Goals.** Fly instruments A and B to obtain information. Analyze information and make predictions. Compare predictions with and without the new technology.
- **Objectives.** Fly instruments using existing satellite and launch vehicle within 24 months. Obtain data for 24 months. Provide comparisons within two weeks of first obtaining data and biweekly thereafter.

Often we see processes that ask for requirements up front. Generally what is meant is that the need, goals, and objectives should be identified and operational concepts developed. It is premature to write requirements until all stakeholders have agreed on the operational concepts.

During the scope discovery process, there are a number of questions that must be answered (see Table 2). It can be beneficial to try to answer these questions initially to understand how much is known and what is unknown. If there are many unknowns, this will be a much more involved and drawn-out process. Significant engineering and analysis efforts, involving conceptual studies, trade studies, modeling, simulation, and prototyping, may be required to obtain the answers. The results of this effort may culminate in modified goals and objectives.

In the preceding example, the operational concept might start with something like the following: We will launch the instruments using an *xyz* class launch vehicle out of the Kennedy Space Center. We will install the instruments in Company A's satellite using the satellite for power, pointing, and communications. We will spend three weeks doing instrument checkout on-orbit and this will include all communications to ground facilities. Following checkout, we will point the instruments per the data-gathering plan and begin taking data. This data will be downlinked daily by the satellite. In the analysis phase, weather forecasters will combine this with other data to make a forecast that will be compared to a forecast made without the instrument data. Both forecasts will be compared to the actual weather to determine the accuracy of the forecasts.

In this example, we have just barely started; however, once the questions in Table 2 are answered we have the next level of questions (see Table 3 on Page 6).

As difficult as gathering this information will be for a single system, it will be magnitudes harder for an SOS. In order for the SOS to succeed, these questions must be answered. The bounds for the SOS must be clearly defined, including operational concepts for each life-cycle phase and all transitions.

The list in Table 3 is a starting point. As you can see from our example operational concept, we have many other questions. Also, this SOS is planned for a four-year period at the least: two years of development and two years of operation. We already know about ground data system changes that are going to need to be accommodated over this time period. If it is successful, we may want to continue using these same instruments for a longer period.

It is not possible to overstress the need for full stakeholder participation and for full life-cycle coverage of operational concepts. Information is needed to feed planning, cost, and schedule estimating, and for developing complete requirements. This scope information must be provided to all stakeholders so they understand and agree to the scope before venturing forward, or the risks will be unmanageable.

Stakeholder buy-in to the SOS scope is essential for a successful SOS and for writing good requirements.

Since one of the characteristics of an SOS can be that of continual evolution, this implies that the scope will also evolve. This does not mean that we can just make it up as we go along. We need answers to the questions in Tables 2 and 3 to begin. We must understand the big picture,

Table 1: *Requirements Basics*

Requirements Basics	
Process	**Benefit**
• Define scope before requirements.	• Bound the problem/solution space.
• Develop operational concepts for the entire life cycle.	• Prevent requirements omissions.
• Identify stakeholders and involve them from the beginning.	• Prevent requirements omissions and misunderstandings.
• Identify external interfaces.	• Ensure the system works within the larger SOS.
• Educate all writers and reviewers on scope.	• Share the vision; prevent misinterpretations.
• Educate all writers and reviewers on what good requirements are.	• Get needed, clear, concise, and unambiguous requirements.
• Capture rationale for each requirement.	• Capture corporate knowledge and limitations imposed by existing systems.
• Capture verification method for each requirement.	• Think ahead to understand how to verify and to ensure verifiableness.
• Validate requirements as they are submitted.	• Reduce review time.
• Ensure each requirement is responsive to the scope.	• Avoid requirement and scope creep.
• Allocate each requirement to the next level.	• Ensure everything is allocated and required.

Table 2: *Questions To Be Answered*

Questions To Be Answered
• What is the initial operational concept for a nominal operation of the SOS?
• How robust does the system need to be in response to off-nominal events?
• What is the high-level functional architecture of the SOS?
• Which systems of the SOS already exist?
• How might the existing systems evolve during the SOS life-cycle?
• How many new systems will need to be developed to meet the need?
• Who are all the stakeholders of the SOS across the entire life-cycle?
• When must the SOS be operational?
• How much money is available?

More Questions To Be Answered
● Who knows what about each existing system?
● What reliable documentation exists for each system?
● Who among the stakeholders can provide added information about their systems?
● How can we contact and work with these stakeholders?
● What are the interfaces that must exist between existing and planned systems?
● Who controls each interface?
● What standards have been used by existing systems that can enable the interfaces?
● What are the standards that may be available for new systems and future systems?
● What are the regulatory requirements that must be met in order to deploy the SOS?
● What changes can we anticipate over the life cycle?
● What evolution in technologies can we expect?

Table 3: *More Questions To Be Answered*

including the environment and the context in which the SOS must exist and operate. We need a plan from which to base our efforts – not just a random effort.

Management

It is not immediately clear after reading many SOS articles who actually manages an SOS effort for different endeavors. Success depends on someone being in charge. There must be an SOS manager, whether or not that manager has any direct control over the individual systems.

Likewise, it seems clear that there must be SOS architects and system engineers to facilitate the efforts for scope and requirements definition, to manage the interfaces, and to provide the sustaining effort essential to an evolving SOS. The organizational affiliations of these people may be many, and they may be selected for their experience with the systems that comprise the SOS.

SOS Requirements

With the scope clearly defined, we can now look toward writing requirements for the SOS. We are doing this with at least a conceptual architecture in mind and with operational concepts that incorporate existing, evolving, and new systems. The requirements for the SOS will reflect capabilities and performance implied by the scope results as well as the limitations and restrictions imposed by existing systems, standards, and regulations.

Allocation

Although requirements will be written to define the capability and performance of the SOS, there is really no such product. Therefore, it is critical that each and every SOS requirement be allocated to an existing or new system or to an interface between two or more systems. The SOS manager is responsible for ensuring the acceptance of allocated requirements by each of the participating system managers (those managers of existing or new systems within the SOS).

There may be situations where we will use a system within the SOS but we will never work with the manager of that system; we will just use what is available as we do with many commercial products today. The SOS system engineers must anticipate and incorporate possible changes to such systems, e.g., Internet or GPS.

Rationale

For system managers to agree to the allocated requirements, they must completely understand each requirement allocated to them and its relationship to the SOS scope. By ensuring that rationale is provided with every requirement, this can be accomplished. In those cases where a requirement is allocated to more than one system, the affected managers must work with their counterparts to define the interfaces.

For example, the rationale might explain how the requirement is constrained by an existing system. The manager of that system can concur that the requirement is correct, or can state that he or she is planning changes that would invalidate the requirement. If the manager never sees the rationale, he or she may assume that the requirement is caused by someone or something else and never acknowledge the planned changes to his or her system.

The requirement will also provide information about how it relates to the scope so that as the scope evolves so will the requirement.

Standards

The number of standards is downright intimidating. Yet use of standards may hold the key to making systems work effectively together in an SOS. It is important to note the impact of standards on the development of cost-effective products. Having standards has enabled the hardware developers to grow and expand while reducing the price of goods to consumers. There are also standards for software, but fewer since software is a much younger field of engineering.

We rely on these standards when we buy a light bulb for our lamp, hook up our new computer to our existing printer cables, or turn on our laptop in airports and hotels around the world. The Institute of Electrical and Electronic Engineers has thousands of standards to help engineers understand what other engineers are talking about [4]. The American National Standards Institute, the ISO [International Organization for Standardization], and the International Electrotechnical Commission are also providers of large numbers of international standards.

Standards are not static and unchanging; they are updated to account for changes in technology and needs. New standards are in development to fix perceived problems. In many areas, using standards is not a common practice, and the design approaches currently being taken will not effectively support an SOS. Thus more emphasis upon standards is essential to successful SOS development.

From the requirements arena, a standard is a requirement that is agreed upon and understood by a wide range of practitioners. Hardware and software designed to interface standards allow each side of the interface to build toward the middle without extensive negotiations and modifications as the two sides evolve. Using standards also enables us to unplug one system and plug in another that also meets the same standard interface.

Defensive and Self-Healing Requirements

One of the biggest examples of SOS complexity exists in our world of computer hardware and software. There are many brands of computers, operating systems, peripheral devices, device drivers, and applications. These all are expected to work together, but problems occur in getting all of these individual systems working together as a SOS.

I have had so many problems with my small personal world of creating a SOS from commercial off-the-shelf products that I cannot understand how anyone keeps large networked systems operational. With hundreds of workstations, networks, routers, etc., how does anyone dare to ever make any updates?

For a robust SOS, if my system and yours communicate or interact in any way, we need to both protect against what can happen at or across the interface. This begins with each of us defining requirements that will protect our integrity regardless of the other's system. We need requirements that defend against problems like those experienced by the car dashboard SOS. We need self-healing requirements for each system to enable its recovery from the impact of other systems.

There seems to be a desire or maybe even a mandate to merge new and existing systems from disparate organizations to achieve a new capability. This is often a long and tedious process and in many cases is simply abandoned in frustration. For example, the National Oceanic and Atmospheric Administration (NOAA), NASA, and the Navy agreed to do a joint project for developing new weather forecasting capability. NASA has had problems just having multiple NASA centers working together on a project, and this program interjected two other government agencies. NASA was responsible for developing the weather instrument. The Navy was responsible for a launch vehicle and a satellite to house the instrument. The NOAA, NASA, and the U.S. Navy all had a say in what the instrument was to measure and responsibility for ground stations to receive the data. A requirement that the instrument data had to be able to be processed by both NOAA and U.S. Air Force ground facilities only added to the program's complexity. NASA was named the project lead.

Attempts were made by the project team to develop operational concepts and write high-level requirements. While the team was able to document a lot of information, they had no formal training in developing operational concepts or writing requirements. This resulted in a system specification that contained very low-level requirements that constrained the instrument design. As the NASA team struggled to write requirements for their instrument, it was clear that they did not know the details for interfacing to the satellite or launch vehicle. These interfaces and the environments expected by and imposed by the other systems are critical to writing the instrument requirements; the Navy was unresponsive to providing the data. With unintentional constraints from above and lack of information, it was impossible to write good instrument requirements.

This is not an uncommon situation; it happens all too often on many government programs.

The end of the saga was that the Navy bailed out and withdrew its support; NASA continued instrument development to interface with an unknown launch vehicle and satellite, and hardware and software now exist. Will it ever be deployed? That is a good question.

Conclusion

Requirements management will be an important aspect of an SOS, as it is to all systems. It is critical to do the prerequirements work of scope definition, documentation, dissemination, and stakeholder buy-in before the SOS requirements are developed. Management of requirements allocation will be a major activity more than in single-system activities. Issues related to allocation, interfaces, and information transfer must be high on management's agenda and resolved swiftly. Extended use of standards and development of standards to empower an SOS may be key to successful endeavors. Each individual system must pay more attention to its defensive and self-healing requirements to participate in future SOS.◆

References

1. Sage, A., and C. Cuppan. "On the Systems Engineering and Management of Systems of Systems and Federations of Systems." Information, Knowledge, and Systems Management 2 (2001): 325-345.
2. Maier, M. Architecting Principles for Systems of Systems. Proc. of the Sixth Annual International Symposium, International Council on Systems Engineering, Boston, MA, 1996: 567-574.
3. Hooks, Ivy F., and Kristin A. Farry. Customer Centered Products – Creating Successful Products Through Smart Requirements Management. New York: Amacom, 2001.
4. Vonderheid, E. "Standards Hidden in Plain Sight." The Institute Mar. 2004.

About the Author

Ivy Hooks is president and chief executive officer of Compliance Automation, Inc., a company whose focus is requirements. Hooks is an internationally recognized expert, author, and speaker on the subject of requirements. She brings experience from a 20-year career at the NASA Johnson Space Center followed by 20 years experience consulting for government and industry.

Compliance Automation, Inc.
1221 South Main ST
STE 204
Boerne, TX 78006
Phone: (830) 249-0308
Fax: (830) 249-0309
E-mail: ivy@compliance automation.com

Coming Events

September 11-17
20th IEEE International Conference on Software Maintenance
Chicago, IL
www.cs.iit.edu/~icsm2004

September 13-16
Embedded Systems Conference
Boston, MA
www.esconline.com/boston

September 20-23
Software Development Best Practices 2004
Boston, MA
www.sdexpo.com

September 20-24
8th International Enterprise Distributed Object Computing Conference
Monterey, CA
www.edocconference.org

September 24-26
Internet, Processing, Systems, and Interdisciplinary Research
IPSI 2004 Stockholm
Stockholm, Sweden
www.internetconferences.net/industrie/stockholm.html

September 27-30
Better Software Conference and Expo
San Jose, CA
www.sqe.com/bettersoftwareconf

October 6-9
Grace Hopper Celebration of Women in Computing
Chicago, IL
www.gracehopper.org

April 18-21, 2005
2005 Systems and Software Technology Conference
Salt Lake City, UT
www.stc-online.org

Applying CMMI to Systems Acquisition

Brian P. Gallagher and Sandy Shrum
Software Engineering Institute

Building on relevant best practices extracted from the Capability Maturity Model® Integration (CMMI®) Framework, the CMMI-Acquisition Module defines effective and efficient practices for government acquisition organizations. Acquisition best practices are focused inside the acquisition organization to ensure the acquisition is conducted effectively, and are focused outside the acquisition organization as it conducts project monitoring and supplier oversight. These best practices provide a foundation for acquisition process discipline and rigor that enables product and service development to be repeatedly executed with high levels of ultimate acquisition success.

The Capability Maturity Model® (CMM®) Integration (CMMI®) has been applied successfully to systems development and maintenance and has helped organizations improve their project management, engineering, and related processes. In the Software Engineering Institute's (SEI) special report "Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results [1]," the following benefits were reported:

- Boeing Australia experienced a 33 percent reduction in the average cost to fix a defect.
- General Motors experienced an 80 percent reduction in late deliveries.
- Lockheed Martin Integrated Systems and Solutions experienced a 30 percent gain in software productivity.

CMM-based process improvement has enabled these organizations to more consistently deliver products and services on time, at high quality, and for the predicted cost.

These gains are not the exception; they are the norm. System development organizations are making great strides in transferring evolutionary capability into their customers' hands. Gains achieved by Department of Defense (DoD) contractors are transferred directly to the fighting men and women of our armed forces as they become more capable and utilize technology faster than ever before. In addition to satisfied customers and a well-equipped warfighter, the return on the investment these organizations have experienced from the implementation of CMMI is substantial. For example, Northrop Grumman [1] enjoyed a 13-to-1 return on investment.

The acquisition process plays a critical role in how the government transfers increased capabilities into operational use. Acquisition professionals must acquire complex systems and systems of systems in order to provide these enhanced capabilities. If using CMMI can help the developers of these systems, why not apply CMMI practices to help the acquirers as well?

CMMI Acquisition Module

In late 2003, a few colleagues familiar with both acquisition practices and CMMI were asked by Mark Schaeffer, principal deputy, Defense Systems, Office of the Under Secretary of Defense (OSD) for Acquisition, Technology, and Logistics (AT&L), to interpret CMMI for use in acquisition organizations. The goal was to publish a streamlined version of CMMI best practices that could easily be implemented through self-improvement and self-assessment activities to help establish effective acquisition practices within acquisition programs. The result was "CMMI-AM [Acquisition Module]" [2], a technical report published by the SEI. Acquisition professionals in government and industry can use this module to improve their processes.

> *"When the supplier's processes are mature, the acquirer with immature processes often encourages short cuts and interferes with the supplier's ability to meet requirements thus adversely affecting quality, cost, and schedule."*

The CMMI *models* and the CMMI *modules* are two different types of products. The CMMI models, which are part of the CMMI Product Suite, are the official documents that contain CMMI best practices, and can be used with a Standard CMMI Appraisal Method for Process Improvement (SCAMPI℠) Class A appraisal to achieve a maturity level.

The CMMI modules, however, are documents that are excerpts from a CMMI model with possible trial additions and are available for piloting and use for process improvement. Modules that are deemed successful may at some time become part of a CMMI model. A module can be used to identify strengths, weaknesses, improvement opportunities, risks, and best practices during an informal gap analysis or as informative material during a benchmarking SCAMPI Class A appraisal using a CMMI model.

Although CMMI contains many best practices that can help an acquisition organization, CMMI-AM provides additional information designed to help acquisition organizations more easily apply CMMI best practices to their processes.

Acquisition Challenges

Systems acquisition is no easy task. If you think about how complex commercial products are, you are seeing just the *tip of the iceberg*. A family car is the result of a complex mix of subcomponents that are engineered into a system. Most DoD weapon and information systems are at least this complex.

Acquirers must not only understand the operational context and codify the desired capabilities or system requirements into something that can be implemented by a development team, but also they must continuously evaluate both the

® Capability Maturity Model, CMM, and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.
℠ CMM Integration, SEI, and SCAMPI are service marks of Carnegie Mellon University.

evolving systems and the development teams' ability to deliver the systems on time and according to requirements, including cost, fit, and function. The acquirer must also identify the risks involved in selecting one development team or set of suppliers over another, and collaborate with them proactively over the life of the program to ensure risks imposed by the acquirer's environment or the operational environment are identified and mitigated.

Unfortunately, the state of acquisition practice is not what it could be. Difficulties abound in government and industry. Increasing complexity of the systems being acquired has overtaken the experience of those acquiring them. Acquisition professionals often do not have the systems engineering or project management experience needed to meet acquisition objectives.

Many acquirers find it difficult to do the following:

- Establish robust systems engineering practices within the program office.
- Stabilize requirements well enough to adequately work with developers/suppliers.
- Estimate the time and effort required for the program to deliver a usable capability or system.
- Enforce schedule milestones and on-time delivery of acquisition products and services.
- Assess the technical risk involved in acquiring particular products from particular suppliers.
- Implement process control measures.
- Track short- and long-term costs in relation to a budget.
- Continuously identify and mitigate risks in a team environment with all relevant stakeholders.

Since the quality of systems is governed largely by the processes used to create and maintain them, improving the processes used by both the acquirer and the supplier will improve the quality of systems. Again, improving the processes of both the acquirer and the supplier is critical. When both have mature and capable processes, the probability of success is highest.

When the acquirer's processes are mature and the supplier's processes are not, the acquirer can mentor the supplier, but the outcome is not predictable. When the supplier's processes are mature, the acquirer with immature processes often encourages short cuts and interferes with the supplier's ability to meet requirements thus adversely affecting quality, cost, and schedule. Acquirers routinely ask contractors to cut systems engineering, quality assurance, and even causal analysis and continuous improvement activities because they fail to see their immediate value to the program.

Many DoD suppliers have a head start on their government customers because they are already using CMMI best practices. To improve the state of acquisition practice, effective acquisition processes must be defined, implemented, measured, and evolved. The contribution of the acquirer must also be more clearly visible as part of program success.

National Defense Authorization Act

The government has shown its desire to improve the state of acquisition practice in Section 804 of the National Defense Authorization Act, released in December 2002 [3]. This section states, "Service/departments shall establish programs to improve the software acquisition process."

The requirements of such a program include the following:

- A documented process for planning, requirements development and management, project management and oversight, and risk management.
- Metrics for performance measurement and continual process improvement.
- A process to ensure adherence to established process and requirements related to software acquisition.

The act also requires that the Office of System Architecture and Investment Analysis (Communications, Command, Control, and Intelligence) and the Undersecretary of Defense AT&L support government programs by the following:

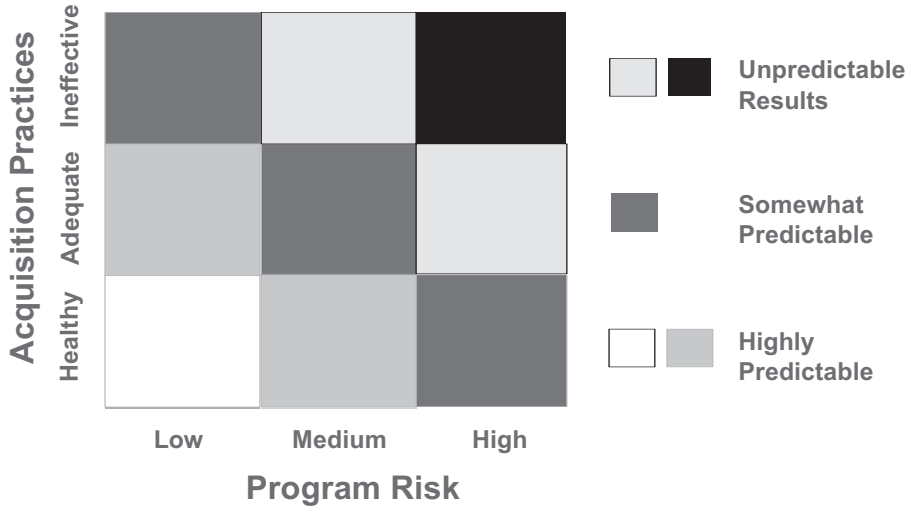
- Prescribe uniform guidance for implementation across the DoD.
- Assist the services and departments by the following:
  - Ensuring that source selection criteria includes past performance and the maturity of the software products offered by potential sources.
  - Serving as a clearinghouse for best practices in software development and acquisition in both the public and private sectors.

This summer, a team of acquisition professionals who are knowledgeable about both CMMI and CMMI-AM has begun a series of pilot appraisals using the module within select DoD programs. In these pilots, participants evaluate the effectiveness of the module in helping program offices establish process improvement programs compliant with Section 804 requirements. This piloting activity is sponsored by Dave Castellano, deputy director, Systems Engineering, Defense Systems, OSD for AT&L.

Managing Acquisition Risk

By improving acquisition processes, acquirers can take on higher-risk programs because they can balance program risk with their improved ability to manage that risk (see Figure 1). The CMMI best practices provide guidance for improving an organization's processes and its ability to manage the development, acquisition, and maintenance of products and product components. The CMMI model and the CMMI-AM assemble best practices into a structure that helps organizations examine the effectiveness of their processes, establish priorities for their improvement, and implement needed improvement.

Figure 1: *Notional Depiction of a Program's Ability to Balance Risk With Healthy Acquisition Practices*



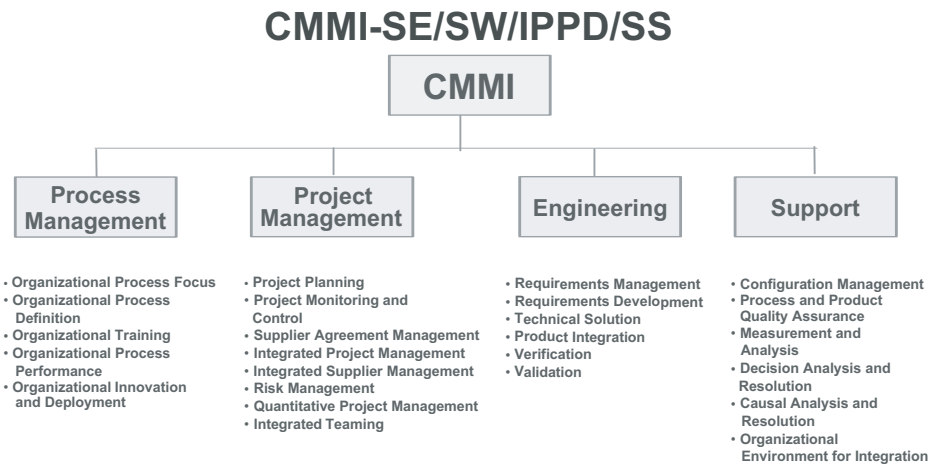


Figure 2: *Structure of CMMI-SE/SW/IPPD/SS Model With a Continuous Representation*

CMMI

CMMI best practices apply to organizations that manage project teams who develop systems (i.e., products and services), not just to the software or systems engineering disciplines within a project team. As illustrated in Figure 2, the CMMI model that includes practices from systems engineering (SE), software engineering (SW), integrated product and process development (IPPD), and supplier sourcing (SS), when used with a continuous representation, organizes the practices into four categories: process management, project management, engineering, and support. This CMMI model was chosen to be used with CMMI-AM because it contains the largest number of best practices that are relevant to the acquisition organization.

Acquisition Best Practices

The CMMI-AM focuses on effective acquisition activities and practices that are implemented by first-level acquisition projects such as a Systems Program Office. Acquisition practices are drawn and summarized from the following sources of best practices:

- CMMI models.
- The Software Acquisition Capability Maturity Model.
- The Federal Aviation Administration Integrated Capability Maturity Model.
- Section 804 of the National Defense Authorization Act.

The CMMI-AM is designed to be used with CMMI best practices as an acquisition lens for interpreting these practices in acquisition environments. Figure 3 illustrates the structure of the module.

Comparing the Module to the Model

If you compare Figures 2 and 3, you will see the difference between CMMI-SE/SW/IPPD/SS and CMMI-AM. Notice that the module does not include the Process Management process areas.

In CMMI-AM Project Management process areas, *Supplier Agreement Management*, *Integrated Supplier Management*, and *Quantitative Project Management* are not transferred from the model. The module adds *Solicitation and Contract Monitoring* as a new process area.

In the Engineering process areas of *CMMI-AM*, *Technical Solution* and *Product Integration* are not transferred from the model.

In the Support process areas of CMMI-AM, *Causal Analysis and Resolution* is not transferred from the model. The module adds *Transition to Operations and Support* as a new process area.

To provide a flavor of CMMI-AM's content, the following includes a best practices' example from one process area within each process area category covered in CMMI-AM.

Project Management

The Project Management process areas included in CMMI-AM are Project Planning, Project Monitoring and Control, Integrated Project Management, Risk Management, Integrated Teaming, and Solicitation and Contract Monitoring.

A few of the best practices included in the Solicitation and Contract Monitoring process area include the following:

- Designate a selection official.
- Establish cost and schedule estimates.
- Evaluate proposals.

Engineering

The Engineering process areas included in CMMI-AM are Requirements Management, Requirements Development, Verification, and Validation.

A few of the best practices included in the Requirements Development process area include the following:

- Establish product and product-component requirements.
- Establish operational concepts and scenarios.
- Analyze requirements to achieve balance.

Support

The Support process areas included in CMMI-AM are Configuration Management, Process and Product Quality Assurance, Measurement and Analysis, Decision Analysis and Resolution, Transition to Operations and Support, and Organizational Environment for Integration.

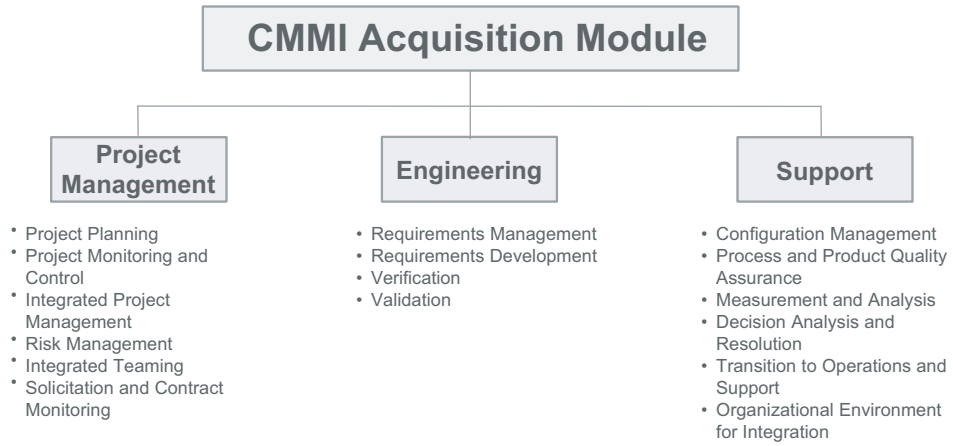
A few of the best practices included in the Transition to Operations and Support process area include the following:

- Establish product transition plans.
- Identify support responsibility.
- Evaluate product readiness.

IPPD Concepts

The fundamental concepts of IPPD incorporated in CMMI-AM include the effective use of cross-functional or multidisciplinary teams, leadership commitment, appropriate allocation and delega-

Figure 3: *The Structure of CMMI-AM*



tion of decision making, and organizational structure that rewards team performance.

Generic Practices

Generic practices ensure that the improvements you make to your processes are effective, repeatable, and lasting. These practices must be considered when implementing the specific practices of the process areas.

Implementing CMMI-Based Process Improvement

To improve acquisition practices, practitioners, projects, and organizations must move from ad hoc acquisition practices to explicit acquisition practices. Using CMMI-AM and the Initiating, Diagnosing, Establishing, Acting, and Learning (IDEAL[SM]) model, a simple improvement process, organizations can do just that (see Figure 4).

Using the IDEAL model and CMMI-AM, a process improvement team would follow each phase in the loop to improve its organization's acquisition practices. The IDEAL model is available at <www.sei.cmu.edu/ideal/ideal.html>.

Where to Go From Here

The CMMI-AM has been going through piloting, and an updated module will be available for use in early Fall 2004. However, there is nothing stopping you from using CMMI-AM now.

To get started, learn as much as you can about CMMI, CMMI-AM, and your organization's acquisition practices. To learn more about CMMI models and CMMI-AM, see <www.sei.cmu.edu/cmmi/models/models.html>. To learn more about CMMI, see <www.sei.cmu.edu/cmmi/>.

Training is available to help you get started, including the Introduction to CMMI training course and CMMI-AM tutorial. There are two types of Introduction to CMMI training available: staged and continuous representations, allowing you to choose the course that is the best fit for your company. Regardless of which course you may take, your choice does not limit your ability to use either or both representations. See <www.sei.cmu.edu/cmmi/training/course-decision.html> for information about selecting an Introduction to CMMI course.

Introduction to CMMI training is available from the SEI or from members of the SEI Partner Network. For more information, refer to <www.sei.cmu.edu/collaborating/partners/partners-tech.html#ICMMI>.

The CMMI-AM tutorial is a one-day introduction to the module designed for acquisition professionals who have attended Introduction to CMMI training and are interested in applying CMMI to acquisition. If you are interested in the CMMI-AM tutorial, contact SEI Customer Relations at <customer-relations@sei.cmu.edu> for more information.

Ensure that your process improvement program has senior management sponsorship and middle management support. Such sponsorship and support is critical to making the program's success possible.

Determine the scope of your initial process improvement program. You can select one or more departments, divisions, programs, or projects. Or, you can select the entire organization. However, it is wise to begin with a smaller scope.

Map your organization's processes to CMMI-AM and CMMI model. It is unlikely that the best practices will map one-to-one with your organization's processes. However, by mapping the existing processes to the practices in CMMI-AM, you will identify where there are gaps. Consider using the IDEAL model to help you implement your process improvement program.

You can conduct an informal gap analysis using CMMI-AM or, if you want a maturity level or capability level rating, you can conduct a benchmarking SCAMPI Class A appraisal using CMMI-SE/SW/IPPD/SS Version 1.1 Continuous with CMMI-AM as additional informative material. If you choose to conduct a SCAMPI Class A appraisal, it will require an SEI-authorized SCAMPI Lead Appraiser. If you do not already have an authorized lead appraiser, there is a list of all currently authorized lead appraisers at <www.sei.cmu.edu/collaborating/partners/partners-tech.html#SCAMPI>. These lead appraisers also have the knowledge to conduct more informal gap analyses using CMMI-AM.

After your gap analysis or appraisal, you will know which processes enable the most useful improvement and the results will guide your process improvement efforts.

Use CMMI-AM as a place to start improving your acquisition processes. You will benefit from the previous experience of successful organizations and

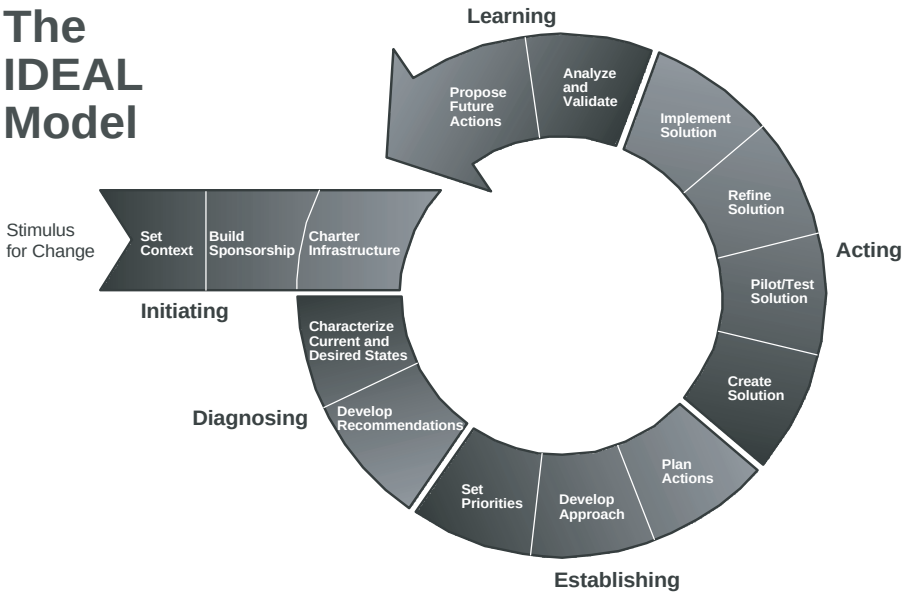


Figure 4: *The IDEAL Model*

> *"Since the quality of systems is governed largely by the processes used to create and maintain them, improving the processes used by both the acquirer and the supplier will improve the quality of systems."*

[SM] IDEAL is a service mark of Carnegie Mellon University.

develop a language that is common among organizations improving their processes – organizations that include the suppliers you work with every day.◆

References

1. Goldenson, Dennis, and Diane Gibson. Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results. CMU/SEI-2003-SR-009. Pittsburgh, PA: Software Engineering Institute, 2003 <www.sei.cmu.edu/publications/ documents/03.reports/03sr009. html>.
2. Bernard, Thomas, Brian Gallagher, Roger Bate, and Hal Wilson. CMMI-AM. CMU/SEI-2004-TR-001. Pittsburgh, PA: Software Engineering Institute, 2004 <www.sei.cmu.edu/ publications/documents/04.reports/ 04tr001.html>.
3. U.S. Congress. "National Defense Authorization Act for Fiscal Year 2002." Calendar No. 163, 107th Congress, 1st Session, S. 1438. Washington, D.C., 2001 <www.theorator. com/bills107/s1438.html>.

Additional Reading

1. CMMI Product Development Team. CMMI for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing Version 1.1 Continuous Representation. CMU/SEI-2002-TR-012. Pittsburgh, PA: Software Engineering Institute, Nov. 2000 <www. sei.cmu.edu/publications/documents /02.reports/02tr011.html>.

About the Authors

Brian P. Gallagher is the director of the Software Engineering Institute's (SEI℠) Acquisition Support Program. He is responsible for building teams from across the SEI's disciplines to support the needs of the Department of Defense and other government agency acquisition programs.

Software Engineering Institute
4500 Fifth AVE
Pittsburgh, PA 15213-3890
Phone: (412) 268-7157
Fax: (412) 268-5758
E-mail: bg@sei.cmu.edu

Sandy Shrum is a senior writer/editor at the Software Engineering Institute (SEI℠). Since 1998, she has been a member of the Capability Maturity Model® Integration (CMMI®) Product Team in roles such as author, reviewer, editor, and quality assurance process owner. Shrum also serves on the CMMI configuration control board and is the CMMI communications manager. She is co-author of the book "CMMI: Guidelines for Process Integration and Product Improvement." Before joining the SEI, Shrum wrote documentation for mainframe- and Unix-based products for Legent Corporation. She has more than 16 years experience as a technical writer in the software industry. Shrum has a Master of Science in professional writing from Carnegie Mellon University and a Bachelor of Science in business administration from Gannon University.

Software Engineering Institute
4500 Fifth AVE
Pittsburgh, PA 15213-3890
Phone: (412) 268-6503
Fax: (412) 268-5758
E-mail: sshrum@sei.cmu.edu

CALL FOR ARTICLES

If your experience or research has produced information that could be useful to others, CROSSTALK can get the word out. We are specifically looking for articles on software-related topics to supplement upcoming theme issues. Below is the submittal schedule for three areas of emphasis we are looking for:

Open Systems/Open Source Software
January 2005
Submission Deadline: August 16, 2004

Risk Management
February 2005
Submission Deadline: September 20, 2004

Personal Computing
March 2005
Submission Deadline: October 18, 2004

Please follow the Author Guidelines for CrossTalk, available on the Internet at <www.stsc.hill.af.mil/crosstalk>. We accept article submissions on all software-related topics at any time, along with Letters to the Editor and BackTalk.

A Recommended Practice for Software Reliability

Dr. Norman F. Schneidewind
Naval Postgraduate School

This article reports on the revisions to the American Institute of Aeronautics and Astronautics' (AIAA) publication "AIAA Recommended Practice for Software Reliability (R-013-1992)" [1]. Sponsored by the AIAA and the Institute of Electrical and Electronics Engineers, the revision addresses reliability prediction through all phases of the software life cycle, since identifying errors early reduces the cost of error correction. Furthermore, there have been advances in modeling and predicting the reliability of networks and distributed systems that are included in the revision.

Software reliability engineering (SRE) is a discipline that can help organizations improve the reliability of products and processes. The American Institute of Aeronautics and Astronautics (AIAA) defines SRE as,

> The application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems. [1]

This recommended practice [1] is a composite of models, tools, and databases, and describes the *what and how* details of SRE, predicting the reliability of software. It provides information necessary for the application of software reliability measurement to a project, lays a foundation for building consistent methods, and establishes the basic principles for collecting the performance data needed to assess software reliability. The document describes how any user may participate in ongoing, software reliability assessments or conduct site- or package-specific studies.

It is important for an organization to have a disciplined process if it is to produce highly reliable software. This article describes the AIAA's recommended practice and how it is enhanced to include the risk to reliability due to requirements changes. A requirements change may induce ambiguity and uncertainty in the development process that cause errors in implementing the changes. Subsequently, these errors propagate through later phases of development and maintenance, possibly resulting in significant risks associated with implementing the requirements. For example, reliability risk (i.e., risk of faults and failures induced by changes in requirements) may be incurred by deficiencies in the process (e.g., lack of precision in requirements).

A revision of the "AIAA Recommended Practice for Software Reliability (R-013-1992)," sponsored by AIAA and the Institute of Electrical and Electronics Engineers, will address reliability prediction through all phases of the software life cycle since identifying errors early reduces the cost of error correction. It will also examine recent advances in modeling and predicting the reliability of networks and distributed systems. At this time, it is not known when this revision will be released. The following sections taken from [1] provide an overview of the planned revisions.

Purpose

The "AIAA Recommended Practice for Software Reliability (R-013-1992)" is used from the start of the requirements phase through the operational-use phase of the software life cycle. It also provides input to the planning process for reliability management.

The practice describes activities and qualities of a software reliability estimation and prediction program. It details a framework that permits risk assessment and predicting software failure rates, recommends a set of models for software reliability estimation and prediction, and specifies mandatory as well as recommended data collection requirements.

The AIAA practice provides a foundation for practitioners and researchers. It supports the need of software practitioners who are confronted with inconsistent methods and varying terminology for reliability estimation and prediction, as well as a plethora of models and data collection methods. It supports researchers by defining common terms, by identifying criteria for model comparison, and by identifying open research problems in the field.

Intended Audience and Benefits

Practitioners (e.g., software developers, software acquisition personnel, technical managers, and quality and reliability personnel) and researchers can use the AIAA practice. Its purpose is to provide a common baseline for discussion and to define a procedure for assessing software reliability. It is assumed that users of this recommended practice have a basic understanding of the software life cycle and statistical concepts.

This recommended practice is intended to support designing, developing, and testing software. This includes software quality and software reliability activities. It also serves as a reference for research on the subject of software reliability. It is applicable to in-house, commercial, and third-party software projects and has been developed to support a systems reliability approach. As illustrated in Figure 1, the AIAA practice considers hardware and, ultimately, systems characteristics.

SRE Applications

Industry practitioners have successfully applied SRE to software projects to do the following [2, 3, 4, 5, 6]:

- Indicate whether a specific, previously applied software process is likely to produce code that satisfies a given software reliability requirement.
- Determine the size and complexity of a software maintenance effort by predicting the software failure rate during the operational phase.
- Provide metrics for process improvement evaluation.
- Assist software safety certification.
- Determine when to release a software system or to stop testing it.
- Predict the occurrence of the next failure for a software system.
- Identify elements in software systems that are leading candidates for redesign to improve reliability.
- Estimate the reliability of a software system in operation using this information to control change to the system.

Figure 1: *System Reliability Characteristics*

Terminology [1]

Software Quality: (1) The totality of features and characteristics of a software product that bear on its ability to satisfy given needs; for example, to conform to specifications. (2) The degree to which software possesses a desired combination of attributes. (3) The degree to which a customer or user perceives that software meets his or her composite expectations. (4) The composite characteristics of software that determine the degree to which the software in use will meet the customer's expectations.

Software Reliability: (1) The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system, as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are encountered. (2) The ability of a program to perform a required function under stated conditions for a stated period of time.

Software Reliability Engineering: The application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems.

Software Reliability Estimation: The application of statistical techniques to observed failure data collected during system testing and operation to assess the reliability of the software.

Software Reliability Model: A mathematical expression that specifies the general form of the software failure process as a function of factors such as fault introduction, fault removal, and the operational environment.

Software Reliability Prediction: A forecast of the reliability of the software based on parameters associated with the software product and its development environment.

The AIAA practice enables software practitioners to make similar determinations for their particular software systems as needed. Special attention should be given in applying this practice to avoid violating the assumptions inherent in modeling techniques. Data acquisition procedures and model selection criteria are provided and discussed to assist in these efforts.

Relationship to Hardware and System Reliability

Hardware Reliability

There are at least two significant differences between software reliability and hardware reliability. First, software does not fatigue, wear out, or burn out. Second, due to the accessibility of software instructions within computer memories, any line of code can contain a fault that, upon execution, is capable of producing a failure. A software reliability model specifies the general form of the dependence of the failure process on the principal factors that affect it: fault introduction, fault removal, and the operational environment.

The failure rate (failures per unit time) of a software system is generally decreasing due to fault identification and removal. At a particular time, it is possible to observe a history of the failure rate of the software. Software reliability modeling is done to estimate the form of the curve of the failure rate by statistically estimating the parameters associated with the selected model. The purpose of this measure is twofold: (1) to estimate the extra execution time required to meet a specified reliability objective, and (2) to identify the expected reliability of the software when the product is released. This procedure is important for cost estimation, resource planning, schedule validation, and quality prediction for software maintenance management.

The creation of software and hardware products is the same in many ways and can be similarly managed throughout design and development. However, while the management techniques may be similar, there are genuine differences between hardware and software. The following are examples:

- Changes to hardware require a series of important and time-consuming steps: capital equipment acquisition, component procurement, fabrication, assembly, inspection, test, and documentation. Changing software is frequently more feasible (although effects of the changes are not always clear) and oftentimes requires only code, testing, and documentation.
- Software has no physical existence. It includes data as well as logic. Any item in a file can be a source of failure.
- Software does not wear out. Furthermore, failures attributable to software faults come without advance warning and often provide no indication they have occurred. Hardware, on the other hand, often provides a period of graceful degradation.
- Software may be more complex than hardware, although exact software copies can be produced, whereas manufacturing limitations affect hardware.
- Repair generally restores hardware to its previous state. Correction of a software fault always changes the software to a new state.
- Redundancy and fault tolerance for hardware are common practice. These concepts are only beginning to be practiced in software.
- Software developments have traditionally made little use of existing components. Hardware is manufactured with standard parts.
- Hardware reliability is expressed in wall clock time. Software reliability is expressed in execution time.
- A high rate of software change can be detrimental to software reliability.

Despite the above differences, hardware and software reliability must be managed as an *integrated* system attribute. However, these differences must be acknowledged and accommodated by the techniques applied to each of these two types of subsystems in reliability analyses.

System Reliability

When integrating software reliability with the system it supports, the characterization of the operational environment is important. The operational environment has three aspects: (1) system configuration, (2) system evolution, and (3) system operational profile.

System configuration is the arrangement of the system's components. Software-based systems are just that; they cannot be pure but must include hardware as well as software components. Distributed systems are a type of system configuration. The purpose of determining the system configuration is twofold:

- To determine how to allocate system reliability to component reliabilities.
- To determine how to combine component reliabilities to establish system reliability.

In modeling software reliability, it is necessary to recognize that systems frequently evolve as they are tested. That is, new code or even new components are added. Special techniques for dealing with evolution are provided in [7].

The system's operational profile characterizes in quantitative fashion how the software will be used. It lists all operations

realized by the software and the probability of occurrence and criticality of each operation.

A system may have multiple operational profiles or operating modes, which usually represent difference in function associated with significant environmental variables. For example, a space vehicle may have ascent, on-orbit, and descent operating modes. Operating modes may be related to time, installation location, customer, or market segment. Reliability can be tracked separately for different modes if they are significant. The only limitation is the extra data collection and cost involved.

Software Reliability Modeling

Software is a complex intellectual product. Inevitably, some errors are made during requirements formulation as well as during designing, coding, and testing the product. The development process for high-quality software includes measures that are intended to discover and correct faults resulting from these errors, including reviews, audits, screening by language-dependent tools, and several levels of test. Managing these errors involves describing, classifying, and modeling the effects of the remaining faults in the delivered product and thereby helping to reduce their number and criticality.

Dealing with faults costs money and impacts development schedules and system performance (through increased use of computer resources such as memory, CPU time, and peripherals requirements). There can be too much as well as too little effort spent dealing with faults. Thus the system engineer (along with management) can use reliability estimation and prediction to understand the current system status and make trade-off decisions.

Prediction Model Validity

In prediction models, validity depends on the availability of operational or test failure data [4]. The premise of most estimation models is that the failure rate is a direct function of the number of faults in the program, and that the failure rate will be reduced (reliability will be increased) as faults are detected and eliminated during test or operations. This premise is reasonable for the typical test environment, and it has been shown to give credible results when correctly applied [3, 5, 6]. However, the results of prediction models will be adversely affected by the following:

- Change in failure criteria.
- Significant changes in the code under test.
- Significant changes in the computing environment.

All of these factors will require a new set of reliability model parameters to be computed. Until these can be established, the effectiveness of the model will be impaired. Estimation of new parameters depends on the measurement of several execution time intervals between failures.

Major changes can occur with respect to several of the above factors when software becomes operational. In the operational environment, the failure rate is a function of the fault content of the program, of the variability of input and computer states, and of software maintenance policies. The latter two factors are under management control and are frequently utilized to achieve an expected or desired range of values for the failure rate or the downtime due to software causes. Examples of management action that decrease the failure rate include avoidance of data combinations that have caused previous failures, and avoidance of high workloads.

Software in the operational environment may not exhibit the reduction in failure rate with execution time that is an implicit assumption in most estimation models. Knowledge of the management policies is therefore essential in selecting a software reliability estimation procedure for the operational environment. Thus, the estimation of operational reliability from data obtained during test may not hold true during operations.

Life-Cycle Approach

A key part of the revision will be the life-cycle approach to SRE. The following example illustrates the life-cycle approach to reliability risk management of the revised recommended practice: This approach has been demonstrated on the space shuttle avionics software [2, 3].

AIAA Practice Applied to the Space Shuttle

The space shuttle avionics software represents a successful integration of many of the computer industry's most advanced software engineering practices and approaches. Since its beginning in the late 1970s, this software development and maintenance project has evolved one of the world's most mature software processes applying the principles of the highest levels of the Software Engineering Institute's Capability Maturity Model®, trusted software methodology, ISO 9001 standards, and [1].

This software process, considered a *best practice* by many software industry organizations, includes state-of-the-practice software reliability engineering methodologies. Life-critical shuttle avionics software produced by this process is recognized to be among the highest quality and highest reliability software in operation in the world. This case study explores the successful use of extremely detailed fault and failure history, throughout the software life cycle, in the application of SRE techniques to gain insight into the flight worthiness of the software and to suggest *where to look* for remaining defects. The role of software reliability models and failure prediction techniques is examined and explained to apply these approaches on other software projects. One of the most important aspects of such an approach is addressed: *how to use and interpret the results* of the application of such techniques.

Interpretation of Software Reliability Predictions

Successful use of statistical modeling in predicting the reliability of a software system requires a thorough understanding of precisely how the resulting predictions are to be interpreted and applied [5]. The primary avionics software subsystem (PASS) (430,000 lines of code) is frequently modified, at the request of NASA, to add or change capabilities using a constantly improving process. Each of these successive PASS versions constitutes an upgrade to the preceding software version. Each new version of the PASS (designated as an operational increment) contains software code that has been carried forward from each of the previous versions (*previous-version subset*) as well as new code generated for that new version (*new-version subset*). By applying a reliability model independently to the code subsets according to the following rules, you can obtain satisfactory composite predictions for the total version:

1. All new code developed for a particular version does use a nearly constant process.
2. All code introduced for the first time for a particular version does, as an aggregate, build up the same *shelf life* and operational execution history.
3. Unless subsequently changed for a newer capability, thereby becoming new code for a later version, all *new code* is only changed thereafter to correct faults.

It is essential to recognize that this approach requires a very accurate code change-history so that every failure can be uniquely attributed to the version in which the defective line(s) of code was first introduced. In this way, it is possible to build a separate failure history for the new code in each release. To apply SRE to your soft-

ware system, you should consider breaking your systems and processes down into smaller elements to which a reliability model can be more accurately applied. Using this approach, the Naval Postgraduate School has been successful in applying SRE to predict the reliability of the PASS for NASA.

Estimating Execution Time

At the Naval Postgraduate School, we estimate execution time of segments of the PASS software by analyzing records of test cases in digital simulations of operational flight scenarios as well as records of actual use in *shuttle* operations. Test case executions are only counted as *operational execution time* for previous-version subsets of the version being tested if the simulation fidelity very closely matches actual operational conditions.

Prerelease test execution time for the new code actually being tested in a version is never counted as operational execution time. We use the failure history and operational execution time history for the new code subset of each version to generate an individual reliability prediction for that new code in each version by separate applications of the reliability model.

This approach places every line of code in the total PASS into one of the subsets of *newly* developed code, whether it is new for the original version or any subsequent version. We then represent the total reliability of the entire software system as that of a composite system of separate components (*new code subsets*), each having an individual execution history and reliability, connected in series. Lockheed Martin is currently using this approach to apply the Schneidewind [8, 9] model as a means of predicting a *conservative lower bound* for the PASS reliability.

Verification and Validation

Software reliability measurement and prediction are useful approaches to verify and validate software. Measurement refers to collecting and analyzing data about the observed reliability of software, for example the occurrence of failures during test. Prediction refers to using a model to forecast future software reliability, for example failure rate during operation. Measurement also provides the failure data that is used to estimate the parameters of reliability models (i.e., make the best fit of the model to the observed failure data).

Once the parameters have been estimated, the model is used to predict the software's future reliability. Verification ensures that the software product, as it exists in a given project phase, satisfies the conditions imposed in the preceding phase (e.g., reliability measurements of safety-critical software components obtained during test conform to reliability specifications made during design) [5]. Validation ensures that the software product, as it exists in a given project phase, which could be the end of the project, satisfies requirements (e.g., software reliability predictions obtained during test correspond to the reliability specified in the requirements) [5].

Reliability Measurements and Predictions

There are a number of reliability measurements and predictions that can be made to verify and validate the software. Among these are *remaining failures*, *maximum failures*, *total test time required to attain a given fraction of remaining failures*, and *time to next failure*. These have been shown to be useful measurements and predictions for: (1) providing confidence that the software has achieved reliability goals, (2) rationalizing how long to test a software component (e.g., testing sufficiently to verify that the measured reliability conforms to design specifications), and (3) analyzing the risk of not achieving *remaining failure* and *time to next failure* goals [6].

Having predictions of the extent to which the software is not fault-free (remaining failures) and whether a failure is likely to occur during a mission (time to next failure) provides criteria for assessing the risk of deploying the software. Furthermore, the fraction of remaining failures can be used as both an *operational quality* goal in predicting total test time requirements and, conversely, as an indicator of operational quality as a function of total test time expended [6].

Risk Assessment

Safety risk pertains to executing the software of a safety-critical system where there is the chance of injury (e.g., astronaut injury or fatality), damage (e.g., destruction of the shuttle), or loss (e.g., loss of the mission) if a serious software failure occurs during a mission. In the case of the shuttle PASS, where the occurrence of even trivial failures is extremely rare, the fraction of those failures that pose any impact to safety or mission success is too small to be statistically significant.

As a result, for this approach to be feasible, all failures (of any severity) over the entire 20-year life of the project have been included in the failure history database for this analysis. Therefore, the risk criterion metrics to be discussed for the shuttle quantify the degree of risk associated with the occurrence of *any* software failure, no matter how insignificant it may be. The approach used can be applied to safety risk where sufficient data exist.

Two criteria for software reliability levels will be defined, then these criteria will be applied to the risk analysis of safety-critical software using the PASS as an example. In the case of the shuttle example, the risk represents the degree to which the occurrence of failures does not meet required reliability levels, regardless of how insignificant the failures may be. Next, a variety of prediction equations that are used in reliability prediction and risk analysis have been defined and included in the document; included is the relationship between *time to next failure* and *reduction in remaining failures*. Then it is shown how the prediction equations can be used to integrate testing with reliability and quality. An example is shown of how the risk analysis and reliability predictions can be used to make decisions about whether the software is ready to deploy; this approach could be used to determine whether a software system is *safe* to deploy.

Criteria for Reliability

If the reliability goal is the reduction of failures of a specified severity to an acceptable level of risk [10], then for software to be ready to deploy, after having been tested for total time (t_t), it must satisfy the following criteria:

Predicted remaining failures

$$r(t\ t) < rc \qquad (1)$$

where,

rc is a specified critical value, and

Predicted time to next failure

$$TF(t\ t) > t\,m \qquad (2)$$

where,

$t\,m$ is mission duration

The total time (t_t) could represent a safe/unsafe criterion, or the time to remove all faults regardless of severity (as used in the shuttle example).

For systems that are tested and operated continuously like the shuttle, t_t, TF (t_t), and t_m are measured in execution time. Note that, as with any methodology for assuring software reliability, there is no guarantee that the expected level will be achieved. Rather, with these criteria, the objective is to reduce the risk of deploying

the software to a *desired* level.

Summary

The existing AIAA practice and planned revisions have been described. The principles of SRE, as applied to the revision have been reviewed. A life-cycle approach to SRE in the revision has been emphasized. The revision is expected to be an important life-cycle software reliability process document to achieve the following objectives:

- Provide high reliability in Department of Defense (DoD) and aerospace safety and mission-critical systems.
- Provide a rational basis for specifying software reliability requirements in DoD acquisitions.
- Improve the management of reliability risk.◆

References

1. American Institute of Aeronautics and Astronautics. AIAA Recommended Practice for Software Reliability (R-013-1992). ISBN: 1563470241. Reston, VA: AIAA, 1992.
2. Billings, C., et al. "Journey to a Mature Software Process." IBM Systems Journal 33.1 (1994): 46-61.
3. Keller, Ted, and N.F. Schneidewind. "Successful Application of Software Reliability Engineering for the NASA Space Shuttle." Software Reliability Engineering Case Studies. International Symposium on Software Reliability Engineering, Albuquerque, N.M., Nov. 1997: 71-82.
4. Musa, J., et al. Software Reliability: Measurement, Prediction, Application. New York: McGraw-Hill, 1987.
5. Schneidewind, N.F., and T. Keller. "Application of Reliability Models to the Space Shuttle." IEEE Software 9.4 (July 1992): 28-33.
6. Schneidewind, N.F. "Reliability Modeling for Safety-Critical Software." IEEE Transactions on Reliability 46.1 (Mar. 1997): 88-98.
7. Musa, J., et al. Software Reliability: Measurement, Prediction, Application. New York: McGraw-Hill, 1987: 166-176.
8. Schneidewind, N.F. "Report on Results of Discriminant Analysis Experiment." 27th Annual NASA/IEEE Software Engineering Workshop, Greenbelt, MD., 5 Dec. 2002.
9. Keller, Ted, N.F. Schneidewind, and P.A. Thornton. Predictions for Increasing Confidence in the Reliability of the Space Shuttle Flight Software. Proc. of the AIAA Computing in Aerospace 10, San Antonio, TX, 28 Mar. 1995: 1-8.
10. Schneidewind, N.F. Reliability and Maintainability of Requirements Changes. Proc. of the International Conference on Software Maintenance, Florence, Italy, 7-9 Nov. 2001: 127-136.

Additional Reading

1. Schneidewind, N.F. "Software Reliability Model With Optimal Selection of Failure Data." IEEE Transactions on Software Engineering 19.11 (Nov. 1993): 1095-1104.
2. Farr, W., and O. Smith. Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide. NAVSWC TR-84-373, Revision 3. Naval Surface Weapons Center, Revised Sept. 1993.
3. Lyu, M. Handbook of Software Reliability Engineering. New York: McGraw-Hill, 1995.
4. Musa, John D. Software Reliability Engineering: More Reliable Software, Faster Development and Testing. New York: McGraw-Hill, 1999.
5. Schneidewind, N.F., and T. Keller. "Application of Reliability Models to the Space Shuttle." IEEE Software 9.4 (Jul. 1992): 28-33.
6. Voas, J., and K. Miller. "Software Testability: The New Verification." IEEE Software 12.3 (May 1995): 17-28.

About the Author

Norman F. Schneidewind, Ph.D., is professor of Information Sciences in the Department of Information Sciences and the Software Engineering Group at the Naval Postgraduate School. Schneidewind is a Fellow of the Institute of Electrical and Electronics Engineers (IEEE), elected in 1992 for "contributions to software measurement models in reliability and metrics, and for leadership in advancing the field of software maintenance." In 2001, he received the IEEE "Reliability Engineer of the Year" award from the IEEE Reliability Society.

Naval Postgraduate School
2822 Raccoon TRL
Pebble Beach, CA 93953
Phone: (831) 656-2719
(831) 372-2144
(831) 375-5450
Fax: (831) 372-0445
E-mail: nschneid@nps.navy.mil

Understanding the Roots of Process Performance Failure

Dr. Robert Charette
ITABHI Corporation

Laura M. Dwinnell
Northrop Grumman IT

John McGarry
U.S. Army Armament Research Development and Engineering Center

The U.S. Department of Defense (DoD) acquisition community seems to be perpetually searching for the answer to the question, "Why isn't program performance significantly improved given all of our investments in process improvement?" Over the past several years, the Office of the Secretary of Defense, in partnership with each of the services, sponsored a performance-oriented assessment effort called the Tri-Service Assessment Initiative that has provided some answers to this question. The initiative was based on a flexible, expert assessment methodology consistently applied to a wide scope of DoD programs. The assessment process allowed for valid cross-program quantification and evaluation of recurring or systemic program issues across the assessed program base. As this systemic analysis capability matured, both DoD program and enterprise managers brought critical analysis questions to the systemic analysis team. One of the most significant of these centered on what the impact of process improvement investments was across the DoD infrastructure. In this article, we will provide a summary of how the results of the DoD cross-program systemic analysis help provide insight into the causes of the recurring process shortfalls in DoD programs.

Despite an increased process focus within Department of Defense (DoD) programs over the past 15 years, there is an increasing gap between program cost, schedule, and technical performance requirements and the capability of program teams to realize them. In our recent analysis of the results of 23 DoD program assessments, *process performance shortfalls* were identified as a primary factor underlying the inability of the programs to meet their acquisition objectives and technical performance requirements. Our analysis showed that nine out of every 10 DoD programs that were assessed exhibited process performance shortfalls – program teams were unable to specify, design, integrate, or execute development processes that met the specific needs of their unique programs. Given the increase in technical and management complexity of future DoD programs, and the trend toward massive systems of systems, our analysis projects that this process-related performance gap will widen.

Performance Assessment and Analysis

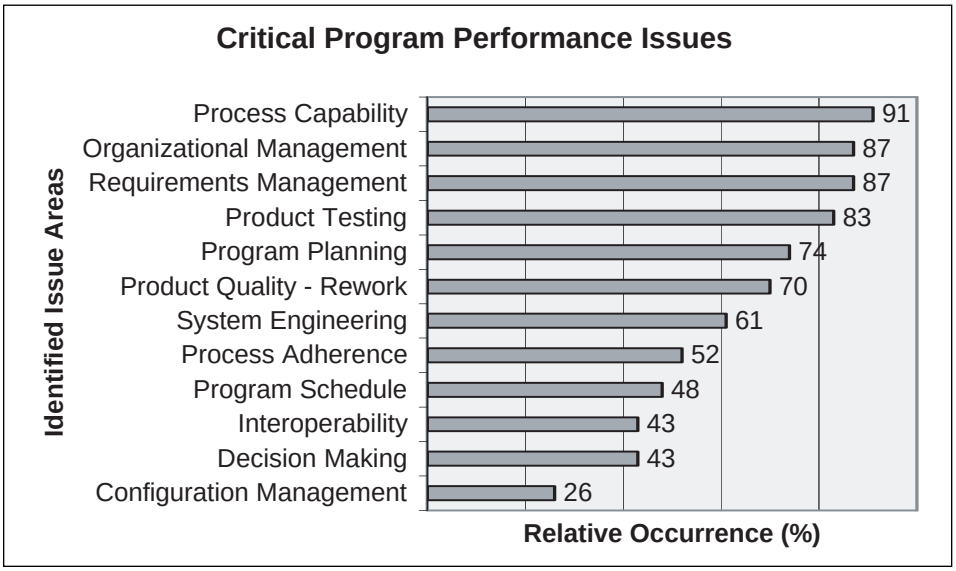
Over the past four years, the Tri-Service Assessment Initiative performed more than 50 major DoD program assessments that spanned the range of acquisition category levels, platforms, domains, and services. This was one of the largest independent assessment programs ever conducted that employed a well-defined and consistent technical approach[1].

The assessment approach encouraged the assessment teams to *drill down* to the causative issues across a very wide scope of acquisition, programmatic, and technical areas, ranging from understanding the general environmental constraints and the customer's agenda to specific contractual, technical requirements, program and project management, and training issues [1]. The assessment approach, with the results delivered to and controlled only by the program manager, also encouraged the assessed program to be open and honest with the assessment teams. This approach, we believe, leads to a truer picture of the state of program performance since the findings are less likely to be *gamed* as in program acquisition oversight audits.

The program performance issues identified by the assessment teams were collected and mapped into a *systemic analysis* database that combined both the quantitative and subjective context data related to the identified performance issues[2]. This analysis approach permitted frequent, relational (cause and effect), and integrated quantitative analysis of the program issues. The results created were realistic, persuasive, and auditable cross-program information that can be effectively used to identify, prioritize, and correct performance shortfalls. Figure 1 provides a relative frequency of occurrence of the types of issues that occurred most often in the assessed programs, issues that materially impacted overall program performance.

Among the recurring issues that were identified, our systemic analysis indicated that the software, systems engineering, and management processes involved in developing and deploying DoD systems were primary contributors to poor pro-

Figure 1: *Critical Program Performance Issues*



gram performance. Process performance issues were of specific concern, and the remainder of this article focuses on our process-related findings.

Process-Related Systemic Analysis Findings

The DoD programs are marked by their complexity and dynamics. The technology embedded in current DoD systems changes both rapidly and repeatedly over the program life cycle. To successfully develop a DoD program requires a highly coordinated team made up of dozens of individual government and contractor organizations that are typically dispersed geographically. The *glue* that holds this complex organization together are the technical and management processes that bring together the technology, resources, knowledge, and skills to execute the program plan. If the appropriate set of processes is not performed, or worse, if the individual processes are inadequate for supporting the program's specific development or evolutionary needs, program success is severely compromised.

A detailed analysis of the program assessment data related to process performance shortfalls led to categorizing the causes of these shortfalls as being related to either *process adherence* or *process capability* (see the sidebar "Process Adherence Versus Process Capability"). The types and relationships of these causative process issues are shown in Figure 2.

It rapidly became clear from our analysis of the systemic issue data that the delivery of adequate *process performance* on any program was directly related both to process adherence (i.e., the ability of an organization to adequately define and implement the technical and management processes required for its programs) and to process capability (i.e., the effectiveness of the defined and implemented organizational processes in meeting a specific program's technical and managerial requirements).

On a positive note, our assessments have not identified any individual programs that are missing the most rudimentary technical or management processes, as shown in the left column of Figure 2. Fifteen years of process improvement efforts have appeared to overcome this one-time common problem. All of the programs that were assessed were well aware of the value of well-defined processes, and of the need to map these processes to the defined business needs within their organizations. Further, most of the organizations assessed were actively involved in a structured process improvement program of some kind.

Our analysis results showed that over 50 percent of the DoD programs that have been assessed have issues involving *process adherence*[4]. This means that the assessments identified performance issues directly related to a program team's ability to implement the technical and management organizational process model or standards that the organization had established as being necessary to ensure program success. The assessment results showed that process adherence shortfalls are most commonly found in the areas of requirements definition, risk management, testing, systems engineering, and technical change management.

As illustrated in Figure 2, our assessment data reveals that there are two general types of process adherence shortfalls. First are the technical or management processes that are *poorly executed*, meaning that they are ineffectively implemented or performed for a particular program. For example, we have found that poor program team communication plagues many programs, largely due to poor implementation of integrated product teams (IPTs) structures within the program. Our analy-

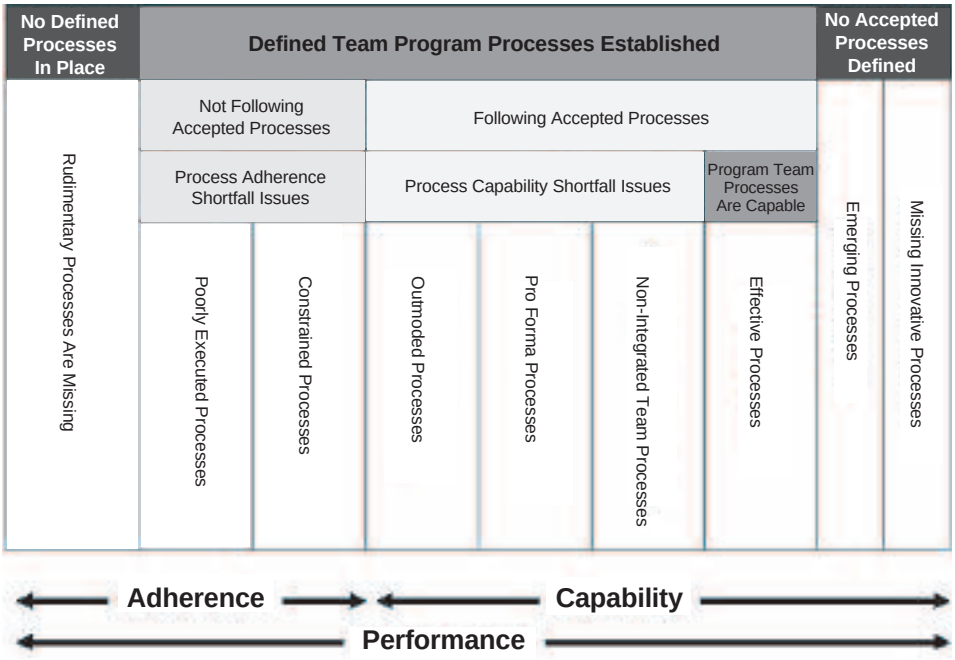
Process Adherence Versus Process Capability

Process performance is the ability to specify, design, integrate, and execute the development processes that meet the specific needs of a unique program. As shown in Figure 2, program process performance is a combination of *both* process adherence and process capability.

Our analysis showed that there are two primary types of process performance shortfalls that impact the overall process performance within a program. The first type of shortfall is related to *process adherence*. Process adherence is defined as the ability of an organization to adequately define and implement the technical and management processes required for its programs. Typically, process adherence adequacy or performance is evaluated against defined process reference models or standards that a parent organization or enterprise has established as being necessary to ensure program success[3]. Common process models include the Software Engineering Institute's Capability Maturity Model® (CMM®), the CMM Integration℠, and ISO [International Organization for Standardization]/International Electrotechnical Commission Standard 15504:1998 for software process assessment. Achievement of a defined maturity level is often viewed as a measure of process adherence for an organization.

The second type of process shortfall relates to *process capability*. Process capability is defined as the effectiveness of the defined and implemented organizational processes in meeting a specific program's technical and management requirements. In general, process capability refers to how well an organization's process models or standards have been adapted and applied to address the specific characteristics and needs of a particular program.

Figure 2: *Types of Technical and Management Process Issues Encountered*



sis further showed that poor risk management and measurement processes were primary causative issues to the IPT problems.

In one program, we discovered that more than 60 IPTs were created, with many of the program team members assigned to six or more individual teams. Furthermore, these IPTs had the responsibility, but not the authority, for making technical decisions (in most cases only recommendations). As one person on the program succinctly put it, "It takes a long time to make a bad decision." We have found that many *best practices* such as IPTs, risk management, or measurement are not being implemented properly on DoD programs, and as a result may cause more problems than they solve.

The second type of process adherence shortfall can be described as *constrained processes*. These are technical or management processes that are not fully implemented or executed because the program team no longer supports or funds them. For instance, we found that the full range of software or systems testing that is planned for at the beginning of a program is often not carried out due to later emerging program budgetary or schedule shortfalls. Testing is in effect traded off against higher-priority program cost or schedule objectives. As a result, errors that should have been discovered during development testing slip into the operational system, causing major problems in the field. One individual on such a program commented, "My worry is not so much whether we deliver on time, but that should the system fail during its operational test, will we be able to tell why?"

Even when program teams were satisfactorily performing the specified organizational team technical and management processes, our analysis showed that the processes themselves were often inadequate to meet the program's performance objectives. In other words, there existed a *process capability* shortfall, indicating that the processes used were ineffective for the situation encountered[5]. As before, several different types of process capability shortfalls have been identified as shown in Figure 2.

The first type of process capability shortfall is the *outmoded process* problem. This occurs when a process model, standard, or practice may no longer be supported, or a specific process-related practice is inappropriate for the situation, e.g., it does not scale for implementation on a large program. While the data showed several instances of these issues, one extreme situation was related to the management of software requirements. In this particular program, the program team was attempting to manage over 20,000 software requirements — *manually*. While the process and related procedures used for requirements was still *theoretically* adequate, it was proving to be extremely labor intensive and error prone. The program had *outgrown* the original process capability. The cost of changing to a new requirements process may have been seen as too expensive and time consuming, so the *outmoded* (and ineffective) process remained in place.

A second type of process capability shortfall is the *pro forma process* approach common to many programs. This occurs when a process is adequately defined but performed in a *check-in-the-box* manner. In other words, the process exists on paper, but no one pays much attention to it. Said another way, there is little value to the output of the process. A common characteristic of pro forma processes is that their outputs are not utilized to make decisions or to improve how the program is being run. Program risk often falls into this category. Risk management is *performed* on most programs, but we found that it is mainly for show. Risks are not communicated and the identified risks frequently do not influence program decision making.

A third type of process capability shortfall identified by our systemic analysis is the *nonintegrated team process*. This occurs when a program team uses several different and often incompatible processes to achieve the same end. This lack of coordination of processes plagues multiple supplier programs where work items are shared. For instance, in one program, because there was a lack of coordinated configuration management processes across the program team, the software product ended up being handled and managed very differently at different times in the development process. This led to major problems on the program as no one could really be certain what version was being used where.

Finally, as shown in Figure 2, there are the processes that are needed for program success, but no accepted practices have been defined. For instance, there is the *emerging process* situation where a new or largely revised process is required, but the program team has failed to define it in sufficient detail. An emerging process does not require adherence to an organizational process standard since the process standard in question may not have been upgraded to include it. For example, many programs appreciate that they have to manage changes in technology over the course of their program development and beyond. However, our assessments have found that many, if not most programs, are managing technological insertion in an *ad hoc* fashion, rather than through any discretely managed process. As a result, technology updates are introduced haphazardly into the development cycle. Since the process for managing technology insertion is defined at the higher Capability Maturity Model® (CMM®) and CMM Integration℠ maturity levels – higher than those usually applied on a DoD program – it is routinely overlooked as being necessary. Additionally, we found that *innovative processes* are required to meet many program's needs and to improve their performance. We found process shortfalls in systems interoperability management, family of systems management, and capability-based acquisition management, among others.

When taken together, process adherence or process capability issues have been found to exist on nine out of every 10 programs assessed. Disturbingly, in 80 percent of the assessed programs where no process adherence issues of merit were found, process capability issues were still discovered. While the program teams are generally aware of the need for improving their adherence to a set of defined processes, the analysis results showed that program team members do not routinely consider their technical and management process capabilities either *individually* or from an *overall program team perspective*. The result is a *program team* process capability and performance shortfall. In short, the full spectrum of a program team's organizational processes are not rigorously evaluated and then tailored to meet the specific characteristics or requirements of the program in question. We expect our results are typical across most DoD programs.

Observations

Our systemic analysis of the recurring program issues led us to several observations about DoD programs and process performance. Our systemic data indicate that new program teams often proceed with processes that are applicable to the previous program they were involved in – not the one they are currently working on. New technology, new policies, new operating environments, etc., pose new process challenges to programs. Unfortunately, these innovative process challenges are often unrecognized until well into a program's development phase – by which time it is too late. The current data suggest that 10 percent to 20 percent

of previously applicable technical or management processes are not appropriate or effective for new program starts. This unrecognized process need, or *process gap*, is especially true in programs where interoperability, systems of systems, family of systems, or network centric warfare requirements are very high.

Second, most adherence-oriented process models or standards are organization-based; they are based on a generalized organizational standard of what *most* projects require, not on what any specific project requires. While these process models are *intended* to be tailored for specific program needs, the data suggest that in practice they often are *not* (see the sidebar "Limitations of Adherence Models"). It appears that many organizations simply apply their standardized, approved corporate process to meet *all* of the diverse programs in their portfolio. Given the high degree of technical and acquisition change that DoD programs face, the inability or unwillingness to adapt defined organizational processes to meet a program's specific characteristics, constraints, and requirements, significant performance shortfalls are almost a given if substantial process tailoring is not done.

Furthermore, evaluations of adherence to a program's process standards are generally made against organizational-based process adherence requirements, not project-specific capability needs. As a result, the evaluation of process adherence can discourage a complete evaluation and tailoring of process standards to meet specific program needs. In other words, bidders on DoD programs end up proposing the use of their corporate or organizational standard processes rather than processes that are tailored to the program they are bidding on. Unfortunately, *one size does not fit all*, and a *best practice* for one program may not work at all for another.

Fourth, there appears to exist a fundamental disconnect between the significance of process adherence and process capability. While process adherence is necessary, it is an *inadequate requirement* for ensuring process performance on a given program. Process adherence is mistakenly seen by too many program teams to automatically equate to process capability. These program teams often do not realize that adherence to a process model equates to real capability only when the process model and the program's technical and management objectives, assumptions, and constraints match extremely well. In a best-case scenario, i.e., optimal program process performance, three items are closely aligned: (1) the specific program's process requirements; (2) the specific implementation of the process model with methods, procedures, and techniques adapted for the program; and (3) the baseline organizational process model or inherent organizational process standard. Since this is rarely the case, there will almost always be a shortfall in program process performance if the process model is not tailored to the situation.

Our assessments also showed that a program team's process capability, as an integrated entity, is rarely considered. A program team's overall process capability does not necessarily equal the sum of the parts of the individual team members. There appears to be little thought given to how the individual processes of the multiple members of a program team may clash or conflict with one another. Just because each program team member may be part of a CMM Level 3 organization does not mean the program team as a whole operates as a Level 3 organization. The program team must recognize early that all of its individual technical and management processes must be tailored first to the specific situation, and then adherence to that tailored process must be enforced. Too many programs reverse the sequence. A program team must measure a project's likelihood of success in relation to *both* process capability and process adherence.

Finally, process integrity is very often reduced due to time, money, or other program pressures. For instance, a program team member may be rated a CMM Level 3 at the beginning of a program, but fall to a Level 2 or 1 by the middle or the end. Similarly, the program team's process maturity may be a Level 3 at program start but it, too, will likely degrade over time. The impact of process degradation is almost never taken into account during program planning, and represents a real threat to program success.

Conclusions

As programs become more complex and as the future military environment becomes more inter-operative, the management and technical process performance required for successful program execution needs to keep pace. From our systemic analysis across recent DoD programs, several conclusions can be drawn:

- Process improvement efforts have overcome the past problem of individual program team members missing rudimentary technical or management processes. However, in all of our assessments, we never encountered a

Limitations of Adherence Models

The Software Engineering Institute's Capability Maturity Model® (CMM®) and CMM Integration℠ have been the favored models against which organizational adherence to software engineering processes are measured. Attaining CMM Level 3 has been the target maturity level DoD programs expect their supplier software development organizations to reach. We have found in our assessments that there is a strong expectation by DoD managers that by achieving CMM Level 3, their software developers (government or contractor) will be equipped to control many if not most of the problems associated with software development on a program.

While setting the CMM Level 3 as a goal to reach has improved software development in DoD programs, it does not guarantee in and of itself that software development on a program will be problem- or risk-free. Many program managers do not understand the limitations of the CMM, and therefore, assume program process performance results that the CMM neither promises nor can deliver.

It is important to remember that the CMM is a *model* aimed at improving an organization's software development process, not the development process of any specific program. The CMM assumes that for an individual program, the organization's standard software process (OSSP) will be tailored to meet the individual program's requirements.

Unfortunately, our assessments have found that tailoring of the OSSP (by which we include the methods/procedures/techniques that implement that process) is often not the case in practice. What usually happens is that the OSSP is used *as is* in a program and little tailoring is performed. This is acceptable if the OSSP and the program-specific software process needs are in close alignment. However, this alignment is unlikely to happen in the general case.

Currently, there is no formal evaluation method that routinely assesses the managerial and technical processes required by the program team as a whole. The data shows that this issue also needs to be addressed if programs are to increase their chances of success.

program where the system was being developed by a single organization. Not only is it now time to focus on process performance rather than just process adherence, but also on *team* process performance as well as individual program team member process performance.

- The DoD program teams must be educated in what process performance means, especially the difference between process adherence – following some repeatable process – and process capability – the true effectiveness of that process in execution. Knowing the difference can be the determining factor between program success and failure.
- The DoD program teams need to evaluate the full spectrum of technical and management process requirements, and then tailor their organizationally based adherence models to meet specific program needs. Careful attention must be given on how to deal with process areas that are outside either the general level of adherence desired or the process adherence model itself.
- The DoD programs should be encouraged to assess their program team's overall process capability. The data suggest that process capability and possibly process adherence be evaluated at request for proposal and at major milestone reviews at the very least to prevent process performance degradation.
- Individual program team members need to collectively ensure that their technical and management processes meet the needs of the program and not necessarily just individual needs.
- The DoD must foster the development of forward-looking, innovative processes and practices that are capable of dealing with the future complexity of DoD acquisitions, developments, and deployments.

Future DoD system complexities will put more pressure on not only software, but also systems engineering and management processes. These processes will need to be more capable, coordinated, and team-integrated. The gap between program expectations and the ability of program teams to produce such systems will continue to grow unless actions are taken to solve the process performance problems in a systemic manner.◆

Reference

1. Baldwin, Kristen, and Laura Dwinnell. "Help Identify and Manage Software and Program Risk." CROSSTALK Nov. 2000: 8-11.

Notes

1. This approach was developed at the Research Development and Engineering Command-Armament Research Development and Engineering Center, Picatinny Arsenal, N.J., and was applied in support of the DoD's Tri-Service Assessment Initiative (TAI). After this article was written, the technical direction of TAI was changed.
2. The results are based upon 23 of the 50 programs assessed. Although over 50 program assessments were conducted, only those that were consistent in terms of issue scope and application of the technical assessment process were included in the systemic analysis program base.
3. These models or standards are designed to meet generic program process requirements, but not the specific process needs of an individual program.
4. This category includes programs with software and other processes that did not meet program team policies or proposed standards, for instance, programs that required CMM Level 3 but the program team was only CMM Level 2.
5. We assume that a process adherence shortfall also translates into a process capability shortfall.

About the Authors

Robert N. Charette, Ph.D., is the president/chief risk officer of the ITABHI Corporation and the director of the Enterprise Risk Management and Governance service for the Cutter Consortium. Charette has worked in all facets of risk management, and has designed and led major international defense and commercial program assessments for over 20 years. Charette was the chief designer of the Tri-Service Assessment Initiative assessment methodology, and was a primary analyst on the systemic analysis team. He is now involved in designing an assessment approach for total program team performance in system-of-systems enterprises.

11609 Stonewall Jackson DR
Spotsylvania, VA 22553-4668
Phone: (540) 972-8150
E-mail: charette@itabhi.com

Laura M. Dwinnell is an information technology employee at Northrup Grumman, specializing in process reengineering and quality improvement. She was a key contributor to the Tri-Service Assessment Initiative and to the systemic analysis model used to perform analysis on the causative issues surrounding Department of Defense program performance shortfalls. Dwinnell has a bachelor's degree in mathematics from George Mason University and a master's degree in operations research and management science.

Northrup Grumman IT
7575 Colshire DR
M/S C6W1
McLean, VA 22102
Phone: (703) 883-8707
Fax: (703) 556-3574
E-mail: laura.dwinnell@ngc.com

John McGarry is the lead engineer for Measurement and Performance Analysis for the Quality Engineering and System Assurance Directorate at the U.S. Army Armament Research Development and Engineering Center (ARDEC). McGarry was the lead architect in the development and application of the Tri-Service Assessment Initiative assessment methodology. He is currently implementing integrated measurement, risk, and assessment technologies in support of government and industry systems development programs under ARDEC's Capability Based Performance Improvement program.

U.S. Army ARDEC
AMSRD-AAR-QES
BLDG 92
Picatinny Arsenal, NJ 07806
E-mail: jmcgarry@pica.army.mil

Software Rejuvenation

Lawrence Bernstein and Dr. Chandra M. R. Kintala
Stevens Institute of Technology

Here is a design approach that makes software more trustworthy, called software rejuvenation. It is a periodic, pre-emptive restart of a running system at a clean internal state that prevents latent faults from becoming future failures. It was used in systems ranging from a Lucent billing unit to NASA's long-duration space mission to Pluto, and is implemented in IBM's Netfinity resource manager. It is easy to apply, uses very little central processing unit time, increases software reliability by two orders of magnitude, and is recommended for all software-intensive systems.

Software modules comprise a large part of life- and mission-critical systems. System crashes are more likely to be the result of a fault in the software than in the hardware. In spite of our best efforts at removing the errors/faults (*bugs*[1]) before deploying those systems, it is wise to assume that bugs remain in the system and those bugs often lead to failures (*crashes*).

Software fault tolerance is aimed at tolerating those residual faults by building mechanisms to watch for failures and recover from them [1, 2]. Fault tolerance is a reactive approach: Failures usually happen at unexpected times, and the built-in mechanisms to recover from those failures will kick-in to restart the system and the service. However, these unscheduled interruptions in service are expensive and can be life-threatening. This article describes a proactive, preventive technique called *software rejuvenation* that prevents faults from becoming failures.

Lawrence Bernstein observed in 1990 that faults/bugs, when triggered in software, do not always cause failures/crashes immediately but take the system into a state where it begins to *decay*[2]. This decay has symptoms of memory leakage, broken pointers, unreleased file locks, numerical error accumulation, etc., causing gradual degradation in availability of service and data quality and eventually leading to a failure/crash.

Based on this observation, a new method to enhance the dependability of a software system, called *software rejuvenation*, was introduced in 1995 by Kintala and his colleagues in Bell Labs [1, 3]. Software rejuvenation is a proactive approach that involves stopping an executing process periodically or when a failure is imminent, cleaning up the internal state of the system, and then restarting it at a known healthy state to prevent a predicted future failure.

Software rejuvenation is as intuitive as occasionally rebooting your PC, except that it was never defined, implemented, modeled, and analyzed for software systems before 1995 [3]. Shari Pfleeger used the term *software rejuvenation* to mean, "…looking back at software work products to try to derive additional information …" in her seminal software engineering book [4]. Her use differs from ours as we focus on the execution of the software during its mission, and she focuses on the software development process.

Use

Since the 1960s, data communication designers knew to have software modules restart a communication line when it hung. Communication line handlers often include retry logic to restart a line if it hangs. IBM implemented these techniques in its data communication systems. Their system network architecture software was especially robust to communication line hangs and restarted lines several times once a hang was detected.

"A billing data collector system, originally built by AT&T and used in most of the U.S. regional telephone companies, was the first system that used software rejuvenation for the entire system and whose use was modeled and analyzed."

An early implementation of this technique was part of the Safeguard Antimissile Missile System software implemented in the 1960s. Software designers noted that hangs could occur once error reporting buffers were full. Rather than clearing the buffers, a simple fix was implemented to restart the lines for the remote launch sites periodically when the system was in a peacetime surveillance mode. This avoided extraneous error reporting and improved the availability of the system. Separate maintenance software monitored the quality of the communication lines.

Software rejuvenation technology became the modern realization of this early design that restarts a line before the hang to avoid potential secondary problems. It is a low-cost, easy-to-implement technology that makes systems more trustworthy in telecommunication systems.

A billing data collector system, originally built by AT&T and used in most of the U.S. regional telephone companies, was the first system that used software rejuvenation for the entire system and whose use was modeled and analyzed [3]. Since then it has been used in many telecommunication applications, transaction processing systems, and Web servers [5]. Billing system failures and the use of software rejuvenation to prevent those failures, as described in [3], are quite similar to the failures and the fix that Nick van der Zweep described recently in *Computer World* [3].

Software rejuvenation is also implemented in IBM's Director Resource Manager [6] for use in applications built on Netfinity cluster systems. Netfinity Director provides an interface to rejuvenate an application using a time interval as well as a prediction based on a number of operating system resource values.

The X2000 computing system for NASA's 15-year long Pluto-Kuiper Express mission has stringent constraints in both performance and dependability. The mission itself has three phases: initial

Cruise phase of 12 years, *Encountering* phase of four months, and *Exploration* phase of three years. The X2000 system has several processor strings, and all their computing power is needed during the critical Encountering phase while only a subset of the strings is required to be in service during Cruise and Exploration phases. This aspect is made use in the X2000 by rotating the individual processor strings to an on-duty and off-duty cycle and rejuvenating the software [7] to increase system reliability.

Recent experiments at Stevens Institute of Technology showed that datalink protocols suffering memory leak failures could be made reliable using rejuvenation libraries without having to fix the memory leak bug [8]. In essence, rejuvenation bounds the execution space for the working software so that latent failure modes are not executed. Had this technology been used in the Patriot Missile system (see the next section) during the first Iraq war, the counter overflow problem causing the anti-scud system to fail would not have occurred.

Patriot Missile Case History

> On Feb. 11, 1991, the Patriot Project Office received Israeli data identifying a 20 percent shift in the Patriot system's radar range gate after the system had been running for eight consecutive hours. This shift was significant because it meant that the target (in this case, the Scud) was no longer in the center of the range gate. The target needs to be in the center of the range gate to ensure the highest probability of tracking the target. The range gate algorithm determines if the Scud is in the Patriot's firing range. If it is, the Patriot fires its missiles.
>
> Patriot Project Office officials said that the Patriot system would not track a Scud when there is a range gate shift of 50 percent or more. Because the shift is directly proportional to time, extrapolating the Israeli data (which indicated a 20 percent shift after eight hours) determined that the range gate would shift 50 percent after about 20 hours of continuous use. Specifically, after about 20 hours, the inaccurate time calculation becomes sufficiently large to cause the radar to look in the wrong place for the target. Consequently, the system fails to track and intercept the Scud.
>
> The range gate's prediction of where the Scud will next appear is a function of the Scud's known velocity and the time of the last radar detection. Velocity is a real number that can be expressed as a whole number and a decimal (e.g., 3750.2563 miles per hour). Time is kept continuously by the system's internal clock in tenths of seconds but is expressed as an integer or whole number (e.g., 32, 33, 34, etc.). The longer the system has been running, the larger the number representing time. To predict where the Scud will next appear, both time and velocity must be expressed as real numbers. Because of the way the Patriot computer performed its calculations and the fact that its registers are only 24 bits long, the conversion of time from an integer to a real number cannot be any more precise than 24 bits. This conversion results in a loss of precision causing a less accurate time calculation. The effect of this inaccuracy on the range gate's calculation is directly proportional to the target's velocity and the length of time the system has been running. Consequently, performing the conversion after the Patriot has been running continuously for extended periods causes the range gate to shift away from the center of the target, making it less likely that the target will be successfully intercepted.
>
> By automatically restoring the registers to a safe initial state every eight hours when there are no targets in track the system can avoid making the fault into a failure. The problem need not be fixed in the algorithm itself. This is precisely the effect of software rejuvenation.
>
> This was not the first time this type of problem caused an ABM [antiballistic missile] system to fail. During the Safeguard Antimissile Test Program conducted at Meck Island in the Kwajalein Atoll, a similar problem occurred in the early 1970s. The test site was in an extended hold due to a range problem. The computers and radars scanned the sky for the target that was still on the launch pad in California. After several hours of idling, the antimissile system computer crashed. A timing register overflowed. The system was not tested in this configuration. The problem was found and fixed and well documented in the Mission Test Reports. Further study led to the innovative idea to restart the computer periodically when it was scanning the sky so that it returned to a known tested state. This design was included in the tactical system design. The design was later applied to avoiding hash table problems in a telephone data switch, and collecting billing data from telephone switches, but unfortunately not in the follow-on Patriot antimissile system. [9]

Figure 1: *Probabilistic State Transition Model for A Without Rejuvenation*

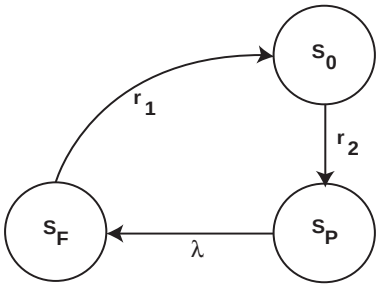
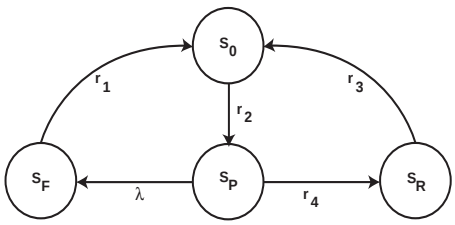


Figure 2: *Probabilistic State Transition Model for A With Rejuvenation*



Modeling and Analysis

Software rejuvenation incurs overhead and should be done at a time when the cost due to service interruption is minimal. Hence modeling the system to find optimal rejuvenation times is crucial. A simple and useful model based on continuous-time Markov chains was first introduced in [3] to analyze software rejuvenation.

Figure 1 shows the model for system A without rejuvenation and Figure 2 is the model for system A with rejuvenation. S_0 is the initial robust state of system A, S_P is the failure probable state, and S_F is the failure state. The transition time from the failed state S_F to robust state S_0 is exponentially distributed with rate r_1 (the repair rate), the transition rate from robust state S_0 to failure probable state S_P is r_2, and λ is that rate for transition from a failure probable state to a failed state. If the system performs rejuvenation, it will go from S_P to S_R at rate r_4 and will transition to

robust state at rate r_3.

From this model you can compute the expected downtime due to rejuvenation over period L to be $(\lambda/r_1+r_4/r_3)/(1+\lambda/r_1+r_4/r_3+(\lambda+r_4)/r_2)$ x L. For example, suppose system A has the following profile:

1. Its mean time between failures (MTBF) is three months; hence, its failure distribution rate λ is 1/MTBF=1/(3x30x24).
2. Its expected repair time is two hours after an unexpected failure, so its repair distribution rate r_1 is (1/2)=0.5.
3. Its expected time to go from robust state to a failure probable state is 10 days; hence, its r_2 is 1/(10x24).
4. Its expected repair time after a scheduled failure is 10 minutes, so its r_3 is (1/(1/6))=6.

The expected downtime of A over a period of one year will then be 7.19 hours without rejuvenation (r_4=0) and 6.36 hours with a rejuvenation frequency of two weeks (r_4=1/(14x24)).

This model was extended using Stochastic Petri Nets to study rejuvenation using the cluster-based fail-over mechanisms in IBM's Netfinity systems [6]. Using this model, it has been shown, for example, that in a two-node cluster system running a database application with one node acting as a spare, the reduction in downtime due to a software rejuvenation interval of 100 hours is 0.74. In the X2000 for the Pluto-Kupier mission, analysis of reliability due to software rejuvenation showed two orders of magnitude improvement and the optimal interval was found to be 31.2 weeks in the 12-year long *Cruise* phase [7].

A number of other modeling techniques were developed to study software rejuvenation in other application scenarios, including the Markov regenerative process model for transaction-based systems, the Weibull distribution model to combine check pointing and rejuvenation, and several others [10].

The Future

Software rejuvenation is ready for industry-wide deployment. It can make software systems more trustworthy. Good designers will use it and move from the state of the art to the state of the practice. It is a *good design practice* for individual systems.

Software rejuvenation is one aspect of self-healing that has gained research interest recently. There are some interesting new problems for software rejuvenation in large-scale, networked, self-healing systems. We describe some of those problems here and make some suggestions:

1. For networked applications, we need to monitor and gather the availability and quality of all the required resources for the application across the network, and then synthesize that gathered data and make a prediction about possible failure of the application or a component in the application. Network application monitoring might be hard to do in such a generalized fashion. You can perhaps do it in a limited domain such as a Voice over Internet Protocol (VoIP) application in an enterprise network.
2. Self-healing systems on a network need alternate paths for communication between components to avoid an impending failure. This may be hard to do in a generalized fashion. But in much the same way as in clustered systems providing redundancy for centralized applications, you can perhaps provide alternate communication paths for some self-healing applications (for example, VoIP) using alternate service provider networks.
3. Modeling and implementation have several problems due to their large-scale nature. What is a state in a large-scale system when *state* is across several products and systems in a network? Perhaps, you need to model the system in a hierarchical, tree-structured fashion decomposing the state into smaller units as you need it for analysis. Failure symptoms are at a system/network (macro) level but rejuvenation actions are at a component (micro) level; how do you correlate the two? This topic is perhaps related to event correlation in network management. How do you do rejuvenation efficiently in very large systems? Perhaps gradual load shedding can be used. What is a safe (clean internal) state to back up to? How do you back up to that state?

"Recent experiments at Stevens Institute of Technology showed that data link protocols suffering memory leak failures could be made reliable using rejuvenation libraries without having to fix the memory leak bug."

Conclusion

Software rejuvenation is a periodic, preemptive restart of a running system at a clean, internal state to prevent future failures. It was used in systems ranging from a Lucent billing unit to NASA's long-duration space mission to Pluto, and is implemented in IBM's Netfinity resource manager. It is one aspect of self-healing systems. Interesting future research directions for software rejuvenation and self-healing are in large-scale networked systems built with commercial off-the-shelf components and open interfaces.◆

References

1. Bernstein, L. "Software Fault Tolerance Forestalls Crashes: To Err Is Human, to Forgive Is Fault Tolerant" in Advances in Computers 58. Highly Dependable Software. Ed. M. Zelkowitz. Academic Press, 2003: 240-285.
2. Lyu, M., Ed. Software Fault Tolerance. New York: John Wiley, 1995.
3. Huang, Y., C. Kintala, N. Kolettis, and N.D. Fulton. Software Rejuvenation: Analysis, Module and Applications. Proc. of 25th Symposium on Fault Tolerant Computing FTCS-25, Pasadena, CA, June 1995: 381-390 <www.ece.stevens-tech.edu/~ckintala/Papers/RejuvFTCS25.pdf>. The Web site <www.software-rejuvenation.com>, maintained by professor Trivedi at Duke University, has a collection of follow-up research papers on the topic.
4. Pfleeger, S.L. Software Engineering Theory and Practice. 2nd ed. Prentice Hall, 2001: 496-502.
5. Li, L., K. Vaidyanathan, and K.S. Trivedi. "An Approach for Estimation of Software Aging in a Web Server." International Symposium on Empirical Software Engineering, Nara, Japan, Oct. 2002.
6. Vaidyanathan, K., R.E. Harper, S.W. Hunter, and K.S. Trivedi. Analysis and Implementation of Software Rejuvenation in Cluster Systems. Proc. of the Joint Intl. Conference on Measurement and Modeling of Computer Systems, ACM SIGMETRICS 2001/ Performance 2001, Cambridge, MA, June 2001.
7. Tai, A.T., L. Alkalai, and S.N. Chau. "Onboard Preventive Maintenance: A Design-Oriented Analytic Study for Long-Life Applications." Performance

Evaluation 35.3-4 (June 1999): 215-232.
8. Bernstein, L., Y.D. Yao, and K. Yao. "Software Rejuvenation: Avoiding Failures Even When There Are Faults." The DoD SoftwareTECH News 6.2 (Oct. 2003): 8-11 <www.softwaretechnews.com>.
9. General Accounting Office. "B-247094, Report to the House of Representatives." Washington, D.C.: GAO, Information Management and Technology Division, 4 Feb. 1992 <www.fas.org/spp/starwars/gao/im92026.htm>.
10. Bao, Y., X. Sun, and K. Trivedi. Adaptive Software Rejuvenation: Degradation Models and Rejuvenation Schemes. Proc. of The International Conference on Dependable Systems and Networks, San Francisco, CA, June 2003.

Notes

1. We use the terms *errors*, *faults*, and *bugs* interchangeably for software systems in this article, even though there are some subtle differences in academic literature.
2. Software decay, sometimes called aging, is not the same as software obsolescence due to changing requirements from the system.
3. Go to <www.computerworld.com> and enter 43636 in QuickLink box, or click on <www.computerworld.com/softwaretopics/software/story/0,10801,88872,00.html>.

About the Authors

Lawrence Bernstein is a professor of Software Engineering at Stevens Institute of Technology. He is a member of the board of the Center for National Software Studies and director of the New Jersey Center for Software Engineering. Bernstein is an expert witness in arbitration cases where he assesses the quality and origins of a large software system. He spent 35 years at Bell Laboratories as chief technical officer managing large software projects. Bernstein holds eight software patents, has given 24 talks, published one book, and has written 58 articles on software engineering. He conceived of the notion of software rejuvenation, encouraged work on studying the dynamic behavior of software, applied and extended software management techniques, and led the work on adopting intermediate level languages in support of military software development.

Stevens Institute of Technology
Hoboken, NJ 07030
Phone: (973) 258-9213
Cell Phone: (862) 485-0814
E-mail: lbernstein@worldnet.att.net

Chandra M.R. Kintala, Ph.D., is a distinguished service professor at Stevens Institute of Technology. Prior to that, he served as vice president of the Network Software Research and Realization Center in Avaya Labs, a spin-off from Bell Labs. Previously, he was director of Distributed Software Research in Bell Labs. Kintala has done pioneering research on Software-implemented Fault Tolerance (SwiFT) for software-implemented fault tolerance and software rejuvenation. He received a *ComputerWorld*-sponsored Smithsonian medal for SwiFT in Lucent in 1998. Under his management, his groups created ExpertNet for enterprise network Voice over Internet Protocol assessment, and Gryphon for network layers 4-7 switch, etc. He has over 40 research publications and five software patents.

Stevens Institute of Technology
Hoboken, NJ 07030
Phone: (908) 580-0991
Cell Phone: (908) 418-7455
E-mail: chandra@kintala.com

Web Sites

INCOSE

www.incose.org

The International Council on Systems Engineering (INCOSE) was formed to develop, nurture, and enhance the interdisciplinary approach and means to enable the realization of successful systems. INCOSE works with industry, academia, and government to disseminate systems engineering knowledge, promote collaboration in systems engineering, establish integrity in systems engineering standards, and encourage research and educational support for systems engineering processes and practices.

Where in Federal Contracting?

www.wifcon.com

Where in Federal Contracting? is a free, noncommercial site that serves the federal and state acquisition and the federal assistance community, including public and private organizations. It provides quick access to acquisition and assistance information such as contract laws and pending legislation, current and proposed regulations, courts and boards of contract appeals, bid protest decisions, contracting newsletters, selected analysis of federal acquisition issues, federal assistance policy, daily listings of grants and cooperative agreements, archived listings of grants and cooperative agreements, and federal assistance sites.

Practical Software and Systems Measurement

www.psmsc.com

Practical Software and Systems Measurement (PSM): A Foundation for Objective Project Management was developed to meet today's software and system technical and management challenges. The Department of Defense and the U.S. Army sponsor PSM. The goal of the project is to provide project managers with the objective information needed to successfully meet cost, schedule, and technical objectives on programs. The PSM is based on actual measurement experience on DoD, government, and industry programs. The PSM supports current software and system acquisition and measurement policy.

Enterprise Composition©

John Wunder
Lockheed Martin Systems Integration

Enterprise information system (EIS) architecture is a system of EISs composed to meet strategic enterprise goals. This composition requires the application of a different set of processes, design patterns, and metrics than those used for stand-alone system architectures. For most enterprise architects, creating EIS architectures can be complicated and fraught with pitfalls, detours, and dead ends. These problems generally are not related to technology but rather caused by misperceptions and culture clash. This article defines a new, agile, incremental approach to EIS architectures and enterprise composition, and shows how it supports the creation and evolution of large EIS architectures such as the Air Force's Global Combat Support System.

Enterprise architects pride themselves on their ability to make stakeholder requirements trade-offs, yet experience shows that there comes a point when the size and complexity of enterprise requirements, especially in nontechnical areas, necessitate extending traditional enterprise system framework approaches (e.g., Department of Defense architecture framework [1], Federal Enterprise Architecture Framework [2], The Open Group Architecture Documentation [3], and Zachman Framework [4]). This article identifies the areas where current enterprise architecture approaches are too rigid or brittle to deal with certain nontechnical and nonfunctional architecture issues associated with architecting (or re-architecting) any large-scale enterprise.

In particular, this article focuses on enterprises with funding, staffing, or political constraints that require new technology/services that replace or must be added to those found in an existing set of applications. This article introduces the term *enterprise composition* to describe a collection of agile processes, metrics, and design patterns that have demonstrated applicability in dealing with these issues.

Gap Analysis: Enterprise IT Lessons Learned

There is an ever-expanding body of knowledge dealing with enterprise architecture frameworks [1, 2, 3, 4] as well as architecture description [5, 6]. Experience has shown that current approaches to enterprise architecture dealing with large-scale enterprises can do the following:

- Lead to unnecessarily rigid designs.
- Require wholesale technology upgrades (i.e., a *big bang*).
- Focus on information technology (IT) cost savings versus process cost savings.
- Result in local optimizations of systems, leading to suboptimal overall enterprise system performance.
- Become bogged down in stakeholder political and cultural *considerations*.
- Rely on traditional metrics such as source lines of code to determine progress.

Table 1 summarizes how enterprise composition addresses some of the shortcomings associated with current approaches to enterprise architecture with respect to large-scale enterprise IT (EIT) systems. The sections that follow will elaborate on lessons learned.

Enterprise Composition Processes

The following sections describe enterprise composition extensions to (1) EIT decision making, (2) EIT framework boundary definition, (3) EIT product selection, and (4) strategic enterprise metric definition.

EIT Decision-Making Process

Martin Fowler [5] recognized that most architecture definitions consist of two elements: (1) breaking the system into parts, and (2) decisions that are hard to change. While it is often the case that enterprise architects consider their architectural decisions to be carved in stone for posterity, when dealing with large enterprise systems, the advice of Gen. George Patton may be more applicable: "A good plan violently executed today is better than a perfect plan executed tomorrow." That is, the composer, while acknowledging that key decisions in structure and policy need to be made, recognizes that making every decision critical, absolute, and perfect, results in bigger risk and higher expense than having a (marginally) less-than-perfect architecture.

From an enterprise composition perspective, composers should apply a customer-centric view following a seven-step process:

1. Define customer goals.
2. Determine how to measure achievement of those goals.
3. Compose a strategic target state that accomplishes those goals.
4. Define the next tactical state on the path to the strategic (i.e., final) state.
5. Assess which of customer's goals will be met in that next incremental implemen-

Table 1: *Comparison of Enterprise Architecture and Enterprise Composition*

Enterprise Architecture Problems	Enterprise Composition Solutions
• Imposes a rigid abstract specification on all aspects of design.	• Establishes a minimum set of flexible interfaces between existing enterprise components.
• Requires mandated modernization efforts just to comply with architecture.	• Focuses on integrating existing capabilities. Modernizations are driven by improved operational processes.
• Primarily justified by cost savings through information technology efficiencies such as enterprise licenses and reduced life-cycle costs.	• Primarily justified by improved higher-level mission processes with IT efficiencies also applicable.
• Is technology-centric with either an Enterprise Resource Planning or a particular commerical off-the-shelf vendor product set as the *Silver Bullet*.	• Is mission-centric and focused above the technology infrastructure.
• Results in agonizingly slow decisions focused on making the *right* choice followed by possible holy wars demanding endless justification of every decision.	• Results in customer-centric decisions based on what works.
• Measures compliance and technology efficiencies through reduction of resources (e.g., systems turned off, reduced operations staff, consolidated hardware and software).	• Measures delivered capabilities and mission efficiencies tied to enterprise metrics (e.g., cost/flying hour, mission capability, kill chain cycle time).

tation of the architecture.
6. Determine how customer goal metrics (to be discussed further in a section that follows) will improve.
7. Commit to those improvements.

While the first three steps are often the easiest, step four is the most important and typically one that many enterprise architects overlook. That is, determining how the enterprise and its existing resources get from their current state to the strategic state (i.e., determining what the most efficient and timely path is to incrementally achieve this [a road map to the] final state, and establishing a process to determine what the next step should be in that direction given the current state and other requirements that have evolved since the last incremental change in the whole EIT). This determination involves steps four through seven.

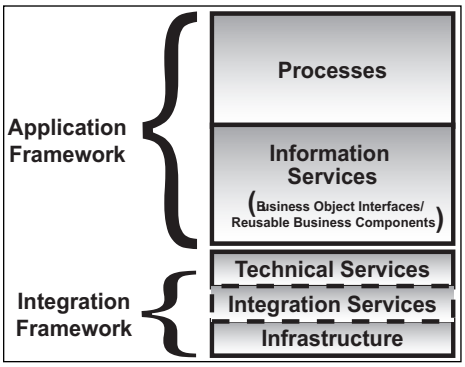
In this way, when the next increment is fielded, its success will be judged not on meeting a date but by measuring how well customer's goals are met. This establishes consistency in the direction of enterprise improvement from increment to increment and through leadership changes.

EIT Framework Boundary Definition Process

As stated previously, composers break the enterprise system architecture into parts. These parts often are organized into a framework within which components providing certain services reside. The Global Combat Support System-Air Force (GCSS-AF) in Figure 1 shows an example of the boundaries in an EIT framework. Enterprise composition guides the composer to minimize the enterprise boundary points to *natural* boundaries and enforce those minimum boundaries rigorously. This insight is the result of the composer following these process steps:

1. Study the problem and solution domain.
2. Correlate the solution domain's technical architecture with existing standards, products, and practices.
3. Define natural boundaries that cleanly separate the EIT into services (see examples in Figure 1).
4. Define objective criteria for boundary implementation.
5. Communicate all boundary information to all enterprise architecture stakeholders.

Figure 1: *GCSS-AF EIS Framework Boundaries*



The GCSS-AF enterprise information system (EIS) [7] shown in Figure 1 has two layered boundaries: the Application Framework and the Integration Framework. The natural dividing line between these layers is the natural separation between Air Force mission information and commercial IT. All Air Force mission-specific information is in the Application Framework, and all generic IT enablers are in the Integration Framework.

Within the GCSS-AF frameworks [8] there are further sub-boundaries or layers. In the Application Framework, the Open Application Group (OAG) Interface Specification [9] provides a natural boundary (or interface) for services upon which components supporting the GCSS-AF Air Force Doctrine 2-4 [10] can be structured. The doctrine creates organizational and information stewardship responsibilities mapped to the OAG standard components such as Inventory, Warehouse, General Ledger, or Budget. In addition, the OAG Interface Specification provides a set of extensible, coarse-grained component boundaries supported by National Institute of Standards and Technology content and syntax tests.

The Application Framework components rely on services provided by the Integration Framework, which relies on standards such as Kerberos, Lightweight Directory Access Protocol V3, Java Authentication and Authorization Service, Public Key Infrastructure, eXtensible Markup Language, HyperText Transfer Protocol, HyperText Markup Language, Web services, Structured Query Language, Portable Operating Systems Interface, Transmission Control Protocol/Internet Protocol, Simple Object Access Protocol, Java 2 Enterprise Edition, or network to create natural security, view, persistence, and messaging boundaries. Objective tests are based on reference implementations of the pertinent standards.

By communicating these boundaries effectively throughout the enterprise, the composer enables the rapid delivery (i.e., composition) of capabilities. This allows implementers to focus within their bounded areas of concern and eliminates the need to address areas outside their particular area of concern. This approach results in a reduction of overall life-cycle cost through reuse of existing services in the GCSS-AF EIT.

EIT Product Selection Process

When enterprise architects address the selection of commercial off-the-shelf products to implement the technical architecture of an enterprise system, they usually start by focusing on each product's capabilities and cost (initial and life-cycle). They conduct extensive trade studies documenting the requirements, weighing the requirements, and assessing the products against those weights.

Often it is the case that, at the end of the evaluation process, the difference between the top products is not statistically significant. Furthermore, a month later the results could change because a new version is released, the chosen product has problems during implementation, or the architect comes to the conclusion that most of the top products could have *done the job* in the first place. Enterprise composition guidelines help the composer improve the product selection process by focusing on a more customer-centric approach rather than a technology-centric approach. This agile and incremental process consists of the following steps:

1. Define the minimum set of mandatory features the customer requires in the product.
2. Determine what existing customer enterprise assets satisfy any of the mandatory features, and allocate them to those assets.
3. Perform high-level, paper, and trade studies on the remaining unsatisfied features using assessments by industry analysts like Gartner, Giga, or Forester to enable a down select to a few products.
4. Instead of taking a technology-centric approach, ask each vendor to provide product compliance levels against the remaining mandatory features. The next step reflects the customer-centric enterprise composition view, as the composer would now ask each vendor to provide at least two reference accounts where existing vendor customers are already using the product in a similar context.
5. Create a survey of the pertinent questions to ask these customer-reference accounts.
6. Set up calls to those customers.
7. Collate the survey results to be used as the prime input to the final selection.
8. Look at the leading candidate product and compare it to the existing personnel skills in the enterprise.
9. If there is a major disconnect between the skill set required to implement the product and the existing skills in the enterprise, then consider the next candidate. The result may be that a less desirable product is preferable because it could be implemented by the enterprise at less cost and risk.

Following this customer-centric, enter-

prise composition-based selection process is usually less expensive than a rigorous, technical, architecture trade-study approach and leads to a product proven to work with built-in expertise from the reference account.

Enterprise Metrics

Architecture metrics have always been a difficult topic to quantify because of their multidimensional nature and lack of good modeling tools. Often these metrics are technology-focused and deal with the performance attributes of the system such as throughput, up time, or even implementation cost. From an enterprise-composition perspective, enterprise architecture metrics measure strategic enterprise goals. In the case of the U.S. Air Force., a set of enterprise productivity measures could include mission capability (aggregate status of the force) and sortie generation capacity. From an enterprise-composition perspective, the metrics chosen are used to show how each increment of the EIT (the addition of new technology/services or mission capabilities) has moved the enterprise closer to the strategic enterprise goals.

Incremental Enterprise Architecture Development Process

Most enterprise architects use the Unified Modeling Language as the design notation to document their architectures [6]. The Unified Software Development Process (USDP) [11] provides a sound, repeatable process model for software development and can be used by enterprise architects to establish the minimum, mandatory artifacts for each increment of the enterprise architecture (e.g., an analysis, tactical, and strategic collaboration diagram would be used to document the goal state and each incremental step).

From an enterprise composition perspective, USDP needs to be extended at both ends of the life cycle. For example on GCSS-AF, the requirements definition phase is proceeded by a business model specification using activity diagrams and use-case diagrams, and the deployment/production phase is extended by using a component repository of XML metadata to facilitate message routing and integration of services.

Enterprise Composition Patterns

The role architecture and design patterns [12] play in enterprise architecture is well recognized [5]. The underlying premise of a design pattern is that,

> ... each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice. [13]

From an enterprise composition perspective, the key patterns that are most useful to the enterprise architect can be labeled as boundary patterns in that they help organize the components and their interfaces so that they form natural boundaries and hide some of the dependencies that otherwise would complicate these interfaces. Following are the boundary patterns discussed in the next sections (sections 2, 3, and 4 are applicable within application framework):

1. Layers Pattern.
2. Canonical/Domain Model Pattern.
3. Model/View/Controller Pattern.
4. Façade Pattern.

Layers Pattern

Usually the Layers pattern is used to define the highest-level boundaries of an EIS. One of the earliest and most widely known examples of the Layers pattern is the seven-layer International Organization for Standardization Reference Model (i.e., Application, Presentation, Session, Transport, Network, Data-Link, and Physical layers). Fowler states that the purpose of layering is "to break apart a complicated software system," [5] giving an architect the following:

1. Intellectual control and understanding within layers.
2. Flexibility to substitute appropriate capabilities at layers.

The number of layers varies according to the area of focus. Fowler advocates three layers [5] (Presentation, Domain, and Data Source). Within GCSS-AF, the EIS is divided into two main layers, or frameworks, which are subdivided into five sub-layers (see Figure 1).

Canonical/Domain Model Pattern

From an enterprise composition perspective, the Canonical/Domain Model pattern can be used to reduce the number of point-to-point interfaces. This allows the architect to select the best tools for his or her job, know the primary interfaces, and only support interfacing to the canonical model decoupling the point-to-point interfaces.

Model/View/Controller Pattern

The Model/View/Controller (MVC) pattern is another long-standing technique used by system designers and architects to separate (via boundary layers) the functionality (the model) from the presentation (the view) through an intermediary interface boundary (the controller) that communicates between component's model and the view.

A derivative of the MVC pattern is the Document View pattern. In this case, the view is dictated by the graphical user interface development tools that link graphical forms with a relational database. The separation of concerns is still maintained between the document/model and the view but the controller function is subsumed within the view function. This is a good pattern for reports and is well supported by Microsoft's Toolset keeping the view synchronized with the record set that typically provides the document or model.

Façade Pattern

The Façade pattern is used to wrap a component in order to simplify its interfaces. A façade can be as simple as an extended script Language Translation script for an XML message or as complicated as an Enterprise Application Integration Extract/Translate/Load tool for a complex, proprietary system interface.

Enterprise Maturity Levels

From an enterprise composition perspective, enterprises that employ EIT mature in a pattern similar to the levels described by the Software Engineering Institute's Capability Maturity Model®. Large, enduring enterprises follow a pattern as they mature. In that pattern, an enterprise determines what governance will provide the most effective support in evolving the EIS. Furthermore, enterprises evolve over long periods of time, and the type and amount of legacy system technology can determine their maturity as well. Using Moore's law as the driving force in the IT industry, the timeline in Figure 2 summarizes the technology shift compared to the number of processors per person.

You should note that most enterprise processes were automated in the 1960s and 1970s when there was little engineering guidance and some severe technology constraints. The client/server era started the shift from mainframe mindset in that most functional areas felt the central enterprise staff was slow and unresponsive, and the central staff felt that the functional depart-

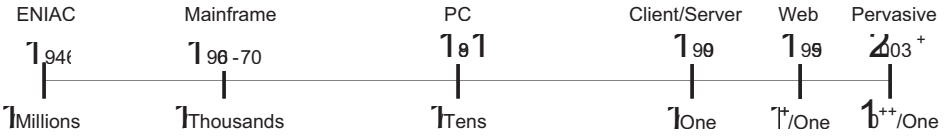


Figure 2: *Timeline of Technology Shift Compared to Processors Per Person*

Maturity Level/ Attributes	Chaotic	Dictatorial	Capability	Optimized
Decision	• Sub-optimal, focused on specific need. • Vendor/Technical criteria. • Looking for silver bullet.	• Sub-optimal, focused on specific need. • Mandated standards. • ERP focus.	• Optimum product selections considering all costs. • Customer-centric approaches. • Core competencies identified and emphasized. • Business case analysis of mandates.	• Driven by optimum enterprise growth. • Members of key industry leadership groups. • Core competencies target predator capabilities.
Artifacts	• Closely coupled throughout. • Holistic deliverables all required capabilities every deliverable. • No separation of layers.	• Layering framework. • Infrastructure administration efficiencies. • Enterprise licenses.	• Everything from Dictatorial. • Canonical Model. • Enterprise Value Chains.	• Information warriors creating own weapons against Canonical Model. • In process measurements mission performance models for continuous improvement.
Metrics	• Meet delivery date.	• IT efficiencies. • Percent Earned Value measurements.	• IT efficiencies. • Earned Value based on complete deliveries work products. • Enterprise Mission Measures.	• Metric capture built-in to Enterprise Value Chains. • Direct measurement of each enterprise contribution.
Rewards	• Subjective assessment.	• Subjective assessment.	• Rewards tied to measured capability delivery.	• Rewards tied to measured capability delivery.

Table 2: *Enterprise Maturity Levels*

ments did not understand the complexities of what they were asking. It was at this point in time that the functional departments took control of their own destiny and within their own control and budgets built the tools that allowed them to respond to mission demands.

These two sets of systems continued to devolve apart along their own paths. The central systems held onto the enterprise applications – such as payroll – while the departments grew department-centric processes starting with simple analysis tools and reports but growing into sophisticated mission critical systems. Soon, with the Web and office tools collecting information from the abundance of individually designed applications, it became clear that the industry had lost control of the information. Today, enterprises are consolidating and trying to get control of their information resources.

Assessment EIT Maturity and Appropriate Governance

Enterprises are trying to regain control of their information flow without restricting the benefits gained from distributed composition. Table 2 details the maturity levels that an enterprise evolves through, and the respective decision characteristics, artifacts delivered, measurements taken, and rewards criteria for success.

Summary

Enterprise composition extends the range of an enterprise architect to allow him or her to address complex, evolving enterprise system architectures. This article has described the processes, metrics, and architectural design patterns that have demonstrated applicability in dealing with these unique challenges.◆

References

1. Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance Architecture Framework, v.2.0. 18 Dec. 1997 <www. afcea. org/education/courses/archfwk2.pdf>.
2. Federal Enterprise Architecture Framework, v.1.1. Sept. 1999 <www.cio.gov/ documents/fedarch1%2Epdf>.
3. The Open Group Architecture Framework <www.opengroup.org/products/ publications/catalog/ar.htm>.
4. Zachman Framework <www.zifa.com>.
5. Fowler, Martin. Patterns of Enterprise Application Architecture. Boston, MA: Pearson Education Inc., Mar. 2003.
6. Clements, Paul, et al. Documenting Software Architectures: Views and Beyond. Addison-Wesley, 2003.
7. U.S. Air Force. Global Combat Support System-Air Force UML Model, 2003 <www.gcss-af.com/cfs/uml>.
8. Global Combat Support System-Air Force <www.gcss-af.com>.
9. Open Applications Group Inc. Open Applications Group Interface Specification v.8. OAGI, 2002 <www.open applications.org>.
10. Cresta, Lt. Col. James. U.S. Air Force, Combat Support Air Force Doctrine Doc. 2-4. U.S. Air Force, 22 Nov. 1999 <www.dtic.mil/doctrine/jel/service _pubs/afd2_4. pdf>.
11. Jacobson, Ivar, Grady Booch, and James Rumbaugh. The Unified Software Development Process. Addison Wesley Longman, Inc. 1999.
12. Gamma, Eric, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
13. Alexander, Christopher, et al. A Pattern Language. New York: Oxford University Press, 1977.

About the Author

John Wunder is a certified Lockheed Martin architect and has been the lead system architect/composer on the Global Combat Support System-Air Force since 1999, and was lead architect for the Dow Chemical Process Control and software architect for the U.S. Army battlefield digitization project. Wunder has been involved in information technology for more than 20 years.

Lockheed Martin
Systems Integration
1801 State RTE 17C
MD 0605
Owego, NY 13827
Phone: (607) 751-6096
Fax: (607) 751-2538
E-mail: john.wunder@lmco.com

Stone Software Development

Sometimes I feel like a procedureless child, a long way from home. Then I remember this marvelous story that my great-grandfather never told me.

Once upon a time in the land of Need-It-Right-Away, there was a great dearth of well-written software. Software customers, greatly desirous of obtaining applications that did exactly what they wanted – yesterday – and at the very least possible cost, had rendered all of their in-house developers into beaten, sniveling hackers, who cowered within fabric-covered holes, ate fat mixed with refined flour and sugars, updated their resumes, and dreamed of better times. Then one day a lone developer from a neighboring province rode into town, strode through the swinging doors of the largest conference room, hung up his spurs, and began to speak with the local project managers as if he planned to stay for a few accounting cycles.

"There's not a charge number to be had in any of our kingdoms," he was told. "Our needs are too urgent, we can't afford to impact any of our schedules by bringing you on board. Besides, unless you can provide a product before COB that is exactly what we are imagining at this very moment for less than minimum wage, why, you're just wasting our time."

"Ah, no problemo" the lone developer replied. "In fact, I was thinking of creating an application to share with all of you based on the stone development method." He pulled an old tempest-hardened laptop from his saddlebag and booted it up. Then he removed a small, smooth, bluish-tinted stone from his pocket and carefully placed it next to his machine.

By now, a flurry of e-mails and flash notes with rumors of a new development method had drawn many customers and in-house developers to the conference room. They scurried for chairs, popped open containers of carbonated caffeine, and brushed the crumbs of deep-fried artificially flavored foods from their sweatshirts. The lone developer closed his eyes, stretched ergonomically, assumed an enigmatic expression, and then wiggled his fingers over the little blue stone by his keyboard while all of the customers in the room beamed their concept of an ideal system at him via mental telepathy.

"Ahhh," he said after several deep cleansing breaths, "I do so much enjoy stone development. Of course," opening one eye to peek at the customers, "stone development with requirements – that's hard to beat."

The local developers gasped, but after a moment one of the customers admitted that he did have, somewhere, a list of specific and fairly well documented requirements that identified the greater portion of what he imagined the ideal application should provide. "Outta sight!" the lone developer exclaimed as he leafed through the pages and placed them next to his stone. "You know, I once worked on a stone development project with requirements and a few plans, and it was simply incredible. After all," he said with a wink at the project managers, "just asking for something by COB is a little vague, don't you think?"

One of the managers looked at his customer, and between the two of them came up with basic information for a schedule and simple quality assurance and configuration management plans. Encouraged by this, several in-house developers began to interact directly with the customer, while the manager used the simple metrics identified to track progress, risks, and costs. Meanwhile the lone developer's fingers flew over the keys of his laptop, swiftly integrating the flood of information that began to pour forth. Design elements, reviews, testing, and acceptance criteria, all clearly traced in a matrix soon resulted in a secure, relational, Section 508-compliant, real time, Web-enabled, embedded application that was a marvel to everyone who used it. It wasn't quite free or completed by COB the first day, but no one seemed to notice.

The customers and managers of the land of Need-It-Right-Away offered the lone developer a prestigious title, a black belt, and a great deal of money for his little blue stone, but he graciously declined and eventually rode off into the sunset. However, history records that from that time forth, there was no longer a lack of well-written software within any of the kingdoms of the land.

And all of the in-house developers, who had taken very careful notes, began to eat better and spend more time with their families.

The End.

— Robert K. Smith
rkensmith@earthlink.net

The Door to the CMMI is Waiting for You ...

Do You Have the Right Keys?

If you want your organization to use common, integrated, and improved processes for both Systems and Software, we can help. The Software Technology Support Center will show your organization how to implement the process improvement methodology of the Capability Maturity Model® Integration℠ (CMMI®), which addresses productivity, performance, costs, and stakeholder satisfaction. Make sure you have the right keys. Call us.

® Capability Maturity Model and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Software Technology Support Center
MASE • 6022 Fir Avenue • Building 1238 • Hill AFB, UT 84056 5820
801 775 5555 • DSN 775 5555 • FAX 801 777 8069 • www.stsc.hill.af.mil

Published by the Software Technology Support Center (STSC)

CrossTalk / MASE
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737